2008

Applications of CSP solving in computer games (camera control)

Mohammed Liakat Ali
University of Windsor

Applications of CSP Solving in Computer Games (Camera Control)

by

Mohammed Liakat Ali

A Thesis
submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2007

Your file Votre référence
ISBN: 978-0-494-42266-3
Our file Notre référence
ISBN: 978-0-494-42266-3

ABSTRACT

While camera control systems of commercial 3D games have improved greatly in recent years, they are not as fully developed as are other game components such as graphics and physics engines. Bourne and Sattar (2006) have proposed a reactive constraint based third person perspective camera control system. We have extended the capability of their system to handle occlusion while following the main character, and have used camera cuts to find appropriate camera positions for a few difficult situations. We have developed a reactive constraint based third person perspective chase camera control system to follow a character in a 3D environment. The camera follows the character from (near) optimal positions defined by a camera profile. The desired values of the height and distance constraints of the camera profile are changed appropriately whenever the character enters a semi-enclosed or an enclosed area, and the desired value of the orientation constraint of the camera profile is changed incrementally whenever the optimal camera view is obstructed. Camera cuts are used whenever the main character backs up to a wall or any other obstructions, or comes out of a semi-enclosed or an enclosed area. Two auxiliary cameras to observe the main camera positions from top and side views have been added. The chase camera control system achieved real-time performance while following the main character in a typical 3D environment, and maintained an optimal view based on a user specified/selected camera profile.

To
my father, Juhurul Haque Bhuiyan
and
my mother, Jamila Aktar Khatun

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Scott Goodwin for countless hours of discussion on the concepts and implementation of the camera control method, advice and guidance to complete this thesis. When I started my graduate study under his supervision, I was barely aware of research on CSPs in computer science and had very limited enthusiasm about computer games. Thus, it was indeed a difficult task to explain to me something in the forefront of the application of AI. I would like to thank the members of my thesis committee, Dr. Jonathan Wu of the Department of Electrical and Computer Engineering and Dr. Ziad Kobti of the School of Computer Science, for their advice and guidance. My appreciation goes to Dr. Christie Eziefe, chair of defence, for her help and support during my studies at the University of Windsor.

I wish to also thank Dr. Owen Bourne for providing me with the source code of one of his camera control systems including the Sliding Octree Solver. Thanks to my fellow students, staff and teachers at the School of Computer Science for their consultations, cooperation, and help.

I am grateful to my wife Salma for her sincere support and encouragement during my undergraduate and graduate studies. Without her help, this thesis work could never have been completed. My sons Rafi and Arko amazed me with their enthusiastic discussions about computer game making from a young user's perspective.

The project was partly supported by an NSERC Post-graduate Scholarship.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

A virtual camera can be described as a window to the 3D virtual world (Bourne 2006). A user observes a virtual 3D world through a virtual camera (Christie et al. 2005).

The idea of perspectives is utilized to implement different kinds of camera control systems. Three kinds of perspectives for camera control are possible: first person, second person and third person.

In a first person perspective camera control, the virtual camera represents a view of the player. The camera is at a fixed position relative to the main character throughout the game play. First person games such as Doom™, Quake™, Halo™, and World War 2 Online™ use this technique. In a second person perspective camera control, the virtual camera represents a view of an antagonist of the player in the game. This is a very difficult perspective to implement. No commercial computer game ever fully implemented the second person perspective camera control system. In a third person perspective, the virtual camera represents a view of a third person. The camera positions are not fixed relative to any object or absolutely in the world coordinate frame. The camera position changes based on the development of the game play at that point of time. Commercial games such as World of Warcraft™, Full Spectrum Warrior™, and Tom Raider™ use this technique. Three shots by three different perspective camera control systems are shown in fig. 1. Many commercial games such as Oblivion™ have both first person and third person perspective camera control systems. The player can choose either one.

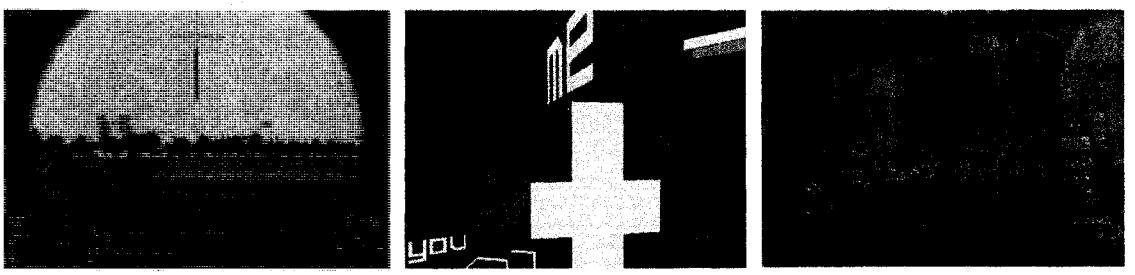

Figure 1: Three Shots by Three Perspective Camera Control Systems.
Shots (from left to right) by a First Person Perspective Camera (Ref: Image_1 2007), a Second Person Perspective Camera (Ref: Image_2 2007), and a Third Person Perspective Camera (Ref: Image_3 2007)

A chase camera (Stone 2004) is a third person perspective camera that keeps the main character in view while maintaining some desired separation. It can also keep one or more additional characters, objects or points of interest in view.

The seven major degrees of freedom (DOF) of a camera are three Cartesian coordinates of its position in a 3D environment, three Euler angles of its orientation, and one angle representing its field of view (FOV). There are other camera control parameters such as aspect ratio of the viewport, depth of field of the camera, shutter speed, and lighting of the environment (Drucker 1994). The camera position and its orientation angles define a camera coordinate frame. In OpenGL, this camera coordinate frame can also be defined by the camera position, the target character position, and an up vector using a GLU utility library function 'gluLookAt()'. Consequently, the seven most commonly used parameters to control a virtual camera are camera position (three parameters), where it is directed (three parameters), and its FOV (one parameter) (Christie et al 2005). If FOV is kept constant and the target character position is known or determined by another system, the seven parameters can be reduced to three as done in both Bourne (2006), and Bourne and Sattar (2006).

A film or cinema consists of multiple scenes, each of which represents a series of events occurring in continuous space and time, and consists of multiple shots. The duration of a shot is from the point at which the camera is turned on until it is turned off (Amerson and Kime 2001). In cinematography, a shot can also be described as a continuous stream of frames, each of which consists of individual images (Bares, Thainimit and McDermtt 2000). Like a film or cinema, a computer game can be logically divided into scenes, shots, frames and images.

To implement scenes and shots, the tasks of the camera control system of 3D computer games can be decomposed into multiple levels following the division of labour of the film-making practices. Hawkins (2003) implements three roles from a film crew: Cinematographer, Editor and Director. The Director determines the availability of good camera shots. The Editor selects shots and decides on the transition between the shots. The Cinematographer finds the camera parameters. Oliveros (2004) implements three levels: low, medium and high, identified as Camera Operator, Cinematographer, and Director, respectively. The Director deals with the automatic selection and sequencing of scenes, based on mood, pace, etc. The Cinematographer deals with various types of shot setup, incorporating photographic and cinematic idioms. The Camera Operator deals with finding camera parameters by locating the values of mainly seven major DOFs of a virtual camera for the selected shot.

A camera system which can enhance viewers' experience significantly must be built on top of an efficient Camera Operator. The Camera Operator should be capable of finding

appropriate camera parameters for the selected shot. In addition, it must have real-time performance.

The motivation behind this thesis is discussed in the next section. In section 1.2, the statement of the problems is provided. In section 1.3, discussion centres on relevant issues. Finally, the outline of the thesis structure is provided in section 1.4.

1.1 Motivation

In interactive computer games, the target character position is not fixed. Camera movements can be based on a prediction of the future character movements or a reaction to the current character movements. In the predictive camera control systems, statistical means are used to estimate target character position for the next frame. With the estimated target position, the next frame is planned, allowing the camera to capture a shot more precisely during fast changes of scenario. Obviously, the frame quality depends on the quality of the prediction. In reactive camera control systems, the target character's current position is used to determine the next frame. Relieved of the burden to calculate the next target position in a reactive system, the real-time performance of the system becomes highly feasible.

In the literature, both the predictive constraint based third person perspective camera control systems such as in Halper, Helbing and Strothotte (2001) and the reactive constraint based third person perspective camera control systems such as in Bourne (2006), and Bourne and Sattar (2006), have been investigated. In addition, predictive

constraint based third person perspective chase camera control systems to follow a character have been investigated (Halper, Helbing and Strothotte 2001). However, reactive constraint based third person perspective chase camera control system to follow a character has not yet been investigated fully.

1.2 Statement of Problems

A third person perspective camera control system provides much more information to the user than a first person perspective camera control system (Christie et al 2005; Lin, Shih and Tsai 2004). It can create much more sophisticated computer games than can be achieved with the first person perspective camera control system, enhancing viewer experience significantly (Lin, Shih and Tsai 2004). However, implementation of an automated third person perspective camera control system in a 3D computer game is a difficult and time consuming task.

A camera should be controlled to fulfill the user's viewing requirements which can be specified as constraints on the camera parameters (Drucker and Zeltzer 1994). Various kinds of constraints are used to describe the relationship between the objects (Badros 1998). Constraints on the camera (with or without constraints on objects of the scene or the image) can define a camera shot. The camera motion, position and orientation must be optimized to deal with a number of hard and soft constraints involving factors such as desired distance from the primary subject or any character of interest, and the size and positioning of the subject in the scene.

To find camera parameters for a camera shot, which fully (or partially) satisfy all the constraints, the problem can be cast as a Constraint Satisfaction Problem (CSP) (Christie et al. 2005). Thus, finding an appropriate shot turns into finding an optimal solution for the CSP instance.

We state that a reactive constraint based third person perspective chase camera that keeps the target character visible at all times can be implemented with real-time performance.

1.3 Discussion

Bourne and Sattar (2006) have introduced a reactive constraint based third person perspective chase camera system which uses a local search based constraint solver called the Sliding Octree Solver. Due to a minimal occlusion handling capability, the camera control system is not able to keep the target character visible all the time. In our study, we extend the Sliding Octree Solver with an occlusion handling capability using ray casting from the character position to the potential camera positions. The character remains visible all the time. If the character is not visible from the optimal camera position, the solver searches for the next best (unobstructed) camera position.

A paper (Ali and Goodwin 2008) based on this thesis work is accepted in the following workshop.

- *1st IEEE International Workshop on Digital Entertainment, Networked Virtual Environments, and Creative Technology – DECT 2008,* to be held along

with the *Fifth Annual IEEE Consumer Communications & Networking Conference – CCNC 2008,* in Las Vegas, NV, USA, January 10-12 2008.

The success of the technique will be judged by the following.

a) Comparing camera shots for a few difficult positions in semi-enclosed/enclosed areas with those of other approaches.
b) Achieving occlusion-free camera shots of the main character all the time.
c) Achieving real-time performance.

1.4 Outline

In chapter 2, we discuss various aspects of CSP, CSP solving techniques, the virtual camera in the context of the OpenGL graphics pipeline, both constraint and non-constraint based camera control systems, and occlusion detection techniques of computer games. In chapter 3, we discuss the experimental design and methodology used to implement the thesis project. An analysis of the results is given in chapter 4. Chapter 5 consists of limitations, conclusions and recommendations for future work. Appendices, References, and Vita Auctoris follow.

CHAPTER 2 REVIEW OF LITERATURE

To implement a reactive constraint based third person perspective chase camera control system for computer games, this thesis covers three areas of study, namely, CSP and CSP solving techniques, camera control in computer games, and occlusion detection techniques. Occlusion detection technique or visibility testing is a part of the collision detection and response system of games. We deal with occlusion detection only, as we are interested only in determining if a character is visible from a potential camera position.

Existing CSP solving techniques, camera control systems in computer games, and occlusion detection techniques in the literature will be discussed in this chapter to provide readers with the necessary background information to understand the issues. We provide mathematical derivations, and theoretical and practical discussions to make all the necessary and relevant information available within this thesis.

In section 2.1, definitions and various aspects of CSPs are discussed. In section 2.2, we discuss and compare various solution techniques of CSP. The virtual camera and its control in the context of OpenGL graphics pipeline is explained in section 2.3. The general and constraint based existing camera control systems are discussed in section 2.4 and 2.5, and occlusion detection methods are discussed in section 2.6.

2.1 Constraint Satisfaction Problem (CSP)

A problem solving algorithm can be classified as a general or a special algorithm. The CSP is a general problem solving algorithm. There are three reasons to study a general algorithm. Firstly, tailor-made special algorithms are costly. Secondly, a slight change in problem specifications would render a special algorithm inapplicable. Finally, a general algorithm can be the basis for development of a special algorithm (Tsang 1993).

CSP can be used to model a wide range of practical problems. A large number of problems in AI and other Computer Science areas such as machine vision, belief maintenance, scheduling, temporal reasoning, graph problems, floor plan design, planning genetic experiments, and satisfiability problems can be considered as special cases of CSP (Nadel 1990; Kumar 1992). Few well-known applications of CSP solution techniques are Map coloring problem, N-queens problem and Car sequencing problem (Tsang 1993).

A CSP is defined as a triple of sets (V, D, C). V, D and C are all finite sets. V is a set of variables $\{x_1, x_2, ...x_n\}$. D can be viewed as a function which maps every variable in V to a set of objects of arbitrary type. If D_{x_i} is the set of objects mapped from x_i by D, D_{x_i} will be called the domain of x_i. The domain D_{x_i} must be a non-empty set. C is a set of constraints on arbitrary subsets of variables in V. C may be an empty set. C will restrict the values that the variables in V can take simultaneously (Tsang 1993; Barták 1998; Russell and Norvig 2002).

There are two representations of constraint specification: extensional and intensional. In an extensional representation, a constraint is specified explicitly by listing all the valid combination of values. In an intensional representation, a constraint is specified implicitly by mathematical equations, logical expressions, etc. (Yang 2004). CSPs having discrete variables with finite domains, such as map-colouring and n-queens problem, are the simplest kind of CSPs. Boolean CSPs are also finite domain CSPs because their variables can only take 'true' and 'false' as values. When discrete variables have infinite domains, constraint language has to be used to describe the constraints because the constraints cannot be enumerated (Russell and Norvig 2002).

Arity of a constraint depends on the number of variables of the constraint. A constraint that restricts the values of a single variable is a unary constraint. Arity of a unary constraint is one. A constraint that relates two variables is a binary constraint. Arity of a binary constraint is two. A non-binary constraint relates more than two variables. Arity of a non-binary constraint is more than two (Russell and Norvig 2002; Huang 2004; Yang 2004).

A CSP with only unary or binary constraints is called a binary CSP. A binary CSP can be visualized as a constraint graph where the set of graph vertices corresponds to the set of CSP variables, and the set of graph edges corresponds to the set of CSP constraints (Russell and Norvig 2002; Huang 2004; Yang 2004). A binary CSP and its constraint graph are shown in fig. 2. The left image of the figure is a map to be coloured. A CSP is defined for the map colouring problem. The CSP and its constraint graph are shown in

the right image. A graph representation of a binary CSP allows the CSP solution techniques to take advantage of graph search algorithms.

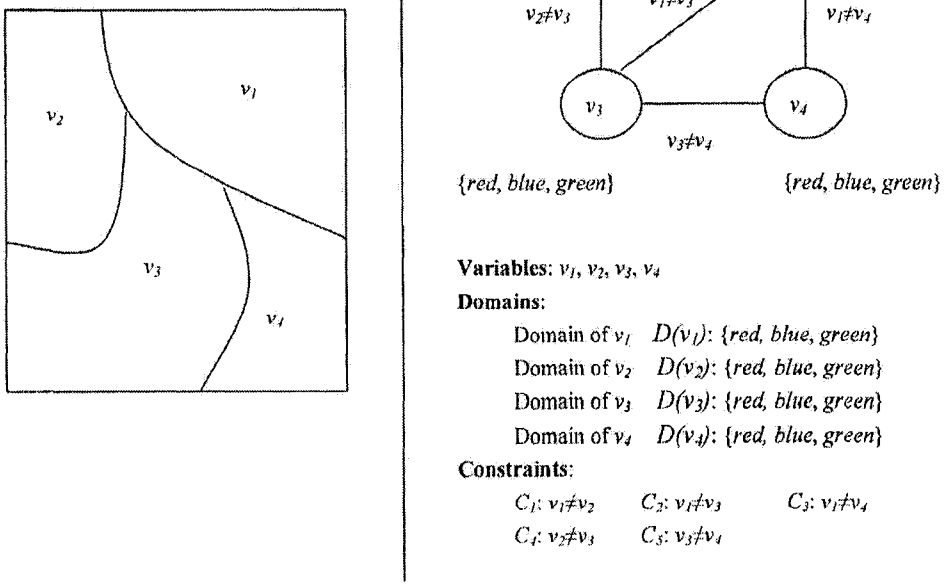

Figure 2: A Binary CSP and its Constraint Graph.
(Ref: fig. 2.1; pp. 7 of Huang 2004 which is adapted from fig. 1.5; pp. 20 of Tsang 1993)

A CSP with any constraint with an arity of more than two is a non-binary or general CSP, and can be visualized as a constraint hypergraph consisting of a set of vertices and a set of hyperedges. A hyperedge allows the connection between more than two vertices (Russell and Norvig 2002; Huang 2004). A non-binary or general CSP is defined in fig. 3. The CSP and its hypergraph are also shown in the figure.

A non-binary or general CSP can be transformed into an equivalent binary CSP which is why the study of binary CSPs becomes so important. Two methods to transform a non-

binary or general CSP are hidden variable encoding and dual encoding (Barták 1998; Huang 2004; Yang 2004).

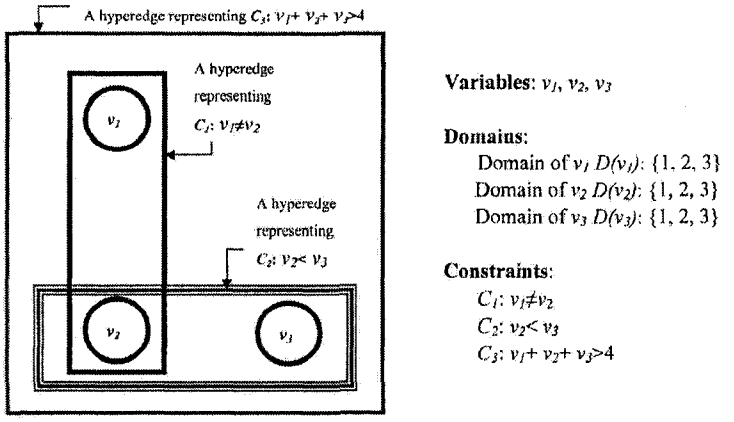

Figure 3: A Non-binary or General CSP and its Hypergraph.
(Ref: fig. 2.2; pp. 10 of Huang 2004)

In hidden variable encoding, the set of variables of the transformed binary CSP consists of the original set of variables of the non-binary or general CSP and a new set of hidden variables. The domains of the original variables remain the same. Each hidden variable represents an original constraint. For a hidden variable, a binary constraint will be added between the hidden variable and each of the original variables involved in the concerned original constraint. The domain of the hidden variable will be unique identifiers for each satisfying tuple of the concerned original constraint (Huang 2004; Yang 2004). Instead of a set of hidden variables, a single hidden variable representing all the constraints can also be used in hidden variable encoding (Barták 1998).

In dual encoding, the set of dual variables of the transformed binary CSP consists of the original set of constraints of the non-binary or general CSP. The domain of each dual

variable will be the set of the tuples that satisfy the corresponding original constraint. The set of the original variable will become the set of new constraints of the transformed binary CSP (Barták 1998; Huang 2004; Yang 2004).

2.2 CSP Solving Techniques

A sound algorithm will return only the solutions of the problem while a complete algorithm will find every solution of the problem. Soundness and completeness are two very desirable properties of an algorithm. To deal with the intractability of some important problems in real life, efficient algorithms, even if incomplete or unsound, are sometimes considered acceptable (Tsang 1993; Russell and Norvig 2002).

There are three unique features of CSP search space. They are identified as: finite search space size, fixed depth of the search tree determined by the numbers of the variable of a CSP, and similarity of sub-trees topology for a particular variable ordering. Due to these characteristics, problem reduction becomes possible (Tsang 1993).

A CSP solving technique may become suitable based on its tightness measure. Tightness of a CSP is defined as the ratio of the number of the solution tuples of the CSP, and the number of distinct tuples consists of all the variables of the CSP (Tsang 1993; Huang 2004; Yang 2004).

During CSP solution, one may search for any solution with or without preference, all the solutions, or an optimal solution based on a cost or object function (Tsang 1993; Barták

1998). CSP solving techniques can be classified as solution synthesis, problem reduction, and search (Tsang 1993).

The solution synthesis technique can be seen as both search and problem reduction. It searches multiple branches simultaneously, and creates constraints for the set of all variables of a CSP in such a way that only solution tuples are present. It generates solution constructively. It will collect a set of legal values starting with a single variable and will generate solution of the CSP incrementally (Tsang 1993).

The node, arc, and path consistency algorithms can be used to reduce search space easily when a CSP is represented as a constraint graph. Node inconsistency can be eliminated by removing all the values from the domain of each variable that does not satisfy the unary constraints of the variable. Once node consistency is achieved for a CSP, all the unary constraints can be dropped from further consideration without affecting the solutions. Arc consistency can be achieved by making each arc of the constraint graph of a CSP consistent (Kumar 1992; Barták 1998). An arc consistency algorithm makes a directed arc (x, y) of a constraint graph consistent by removing all the values from the domain of x for which there is no corresponding value in the domain of y to satisfy the binary constraint that relates x and y. Arc consistency algorithms such as AC-1, AC-2, AC-3, AC-4, AC-5, AC-6 and AC-7 are developed to remove arc inconsistency. AC-3 and AC-4 are the most frequently used arc consistency algorithms (Barták 1998). The concept of arc consistency is generalized as k-consistency because arc consistency is not enough to avoid backtracking. By definition, k-consistency involves two or more arcs.

When k is equal to three, the k-consistency is called a path consistency (Kumar 1992; Tsang 1993; Barták 1998; Russell and Norvig 2002).

Problem reduction techniques remove redundant values from the domain of the variables and tighten the constraints so that fewer assignments satisfy them. The technique is used with the expectation that the reduced problem will become easier to solve (Tsang 1993). However, problem reduction techniques are rarely used alone to find solutions of a CSP. Problem reduction techniques are frequently used along with either solution synthesis or search.

However, most of the CSP solvers find solutions using search. Variables are instantiated with values by the solver, which then checks if the current instantiation is a consistence assignment. In the case of consistent instantiation, it will continue instantiation until a solution is found or lead to an inconsistent assignment (Bourne 2006). In a complete assignment, all the variables of a CSP are instantiated. A complete assignment which satisfies all the constraints simultaneously is a solution of a CSP.

The generate-and-test (GT) method is the simplest method to find a solution of a CSP. In GT, all the possible combinations of the variables are generated and tested. The number of combinations generated and tested is equal to the Cartesian product of all the variable domains. The first combination which satisfies all the constraints is the first solution of a CSP (Kumar 1992). All combinations which satisfy all the constraints are solutions of a CSP. The cost of using the GT method to find the first solution or all the solutions is

enormous. In GT, constraint checking is done after instantiation of all the variables of a complete assignment.

In the backtracking (BT) method, the variables are instantiated one by one. The validity of a constraint is checked whenever all the relevant variables of a constraint are instantiated. In other words, BT performs a depth-first search (Kumar 1992). If any partial or complete assignment violates any constraint, this method backtracks to the most recently instantiated variable. In the case of a constraint violation by a partial assignment, BT will eliminate a subspace from its search space (Kumar 1992; Barták 1998).

Though BT is more efficient than GT, BT has exponential complexity. There are three major drawbacks of BT: late conflict detection, redundant work and thrashing (Kumar 1992; Barták 1998).

BT detects conflict during constraint checking. It cannot detect a conflict before the conflict actually occurs. Advance checking of possible conflicts before instantiation of a variable will eliminate the late detection problem of BT, something accomplish by the Forward Checking (FC) algorithm (Barták 1998).

Three techniques to avoid redundant work and improve performance of the BT algorithm are identified as constraint propagation at search node, intelligent backtracking, and ordering of variable instantiation and instantiation of different values (Kumar 1992).

Using the constraint propagation technique, a given CSP can be transformed into another CSP which has a smaller search space compared to the original one in such a way that the same failures are not encountered again. Parameterized arc consistency procedures can be combined with tree search algorithms. As a result, arc consistency processing can be done on the sub problems rooted at each individual search tree node. However, all the arc consistencies of hybrid algorithms do not achieve full arc consistency at the tree nodes. In the spirit of full arc consistency algorithms such as AC-1, AC-2, and AC-3, the partial arc consistency algorithms are named as AC $\frac{1}{5}$, AC¼, AC⅓, and AC½ based on the degree of partial arc consistency achieved by the hybrid algorithms (Nadel 1990). Many well-known CSP solving algorithms listed in table 1 can be described as a combination of tree search and constraint propagation (Nadel 1990; Kumar 1992).

Table 1: Algorithms as a Combination of Tree Search and Arc Consistency. (Ref: fig. 6, pp. 12 of Kumar 1992 which is adapted from Nadel 1988)

BT = GT + AC $\frac{1}{5}$
FC = GT + AC ¼
Partial Lookahead (PL) = FC + AC ⅓
Full Lookahead (FL) = FC + AC ½
Really Full Lookahead (RFL) = FC + AC

When a branch of search fails, BT will revisit the most recent decision points while an intelligent backtracking algorithm will revisit the set of variables that caused the failure which is called conflict set. The backjumping (BJ) algorithm will revisit the most recently added variable of the conflict set (Russell and Norvig 2002). The dependency-directed backtracking (DDBT) algorithm used in the truth maintenance system and its many simplified implementations, determines the culprit of constraint violation to revisit. Truth

maintenance methods deal with developing general techniques to attach justification or assumptions to the inference. DDBT, learning nogood compound labels (LNCL), backchecking (BC), and backmarking (BM), utilize similarity of search space sub-trees (Tang 1993). Using these techniques, information about a failure is updated and used during the search of the remaining search space (Kumar 1992).

During CSP solving for any one solution without preference, the order of variables to be considered and the order of value instantiation have great implications to the efficiency of the search. Higher search efficiency could be achieved using heuristics. The most common heuristics for variable ordering is based on the 'first-fail' principle where the variable with the fewest remaining alternative values will be selected next for instantiation (Barták 1998). The application of the variable ordering technique will move failures to upper levels of the search tree (Kumar 1992). The most common heuristics for value instantiation ordering is based on the 'succeed first' principle where the value which is most likely to succeed will be selected during instantiation (Barták 1998). The application of value instantiation ordering techniques will move a solution of the CSP to the left of the search tree (Kumar 1992).

Trashing is defined the as repeated failure of the search in different parts of the search space due to the same reason. Node and arc inconsistency of the constraint graph are two causes of thrashing. Trashing can be avoided by making all nodes and arcs of the constraint graph of the CSP consistent (Kumar 1992; Barták 1998). Thrashing of a BT algorithm can be eliminated with much more cost than that incurred by the simple BT if

any of constraint propagation, intelligent backtracking, or variable ordering or value instantiation ordering techniques is applied to an extreme. On the other hand, a simplified version of the techniques can be used together with BT to reduce the overall search space efficiently, but the optimal combination of these techniques will be problem dependent (Kumar 1992).

A CSP is defined as over-constrained if there is no instantiation of the variables that simultaneously satisfy all the constraints. To solve over-constrained CSPs, the concept of classic CSP is extended as Fuzzy CSP, Probabilistic CSP, Weighted CSP and Partial CSP. In addition, constraint hierarchy is used to solve an over-constrained CSP. Constraints can be specified with hierarchical preference. Solutions will satisfy mostly preferred constraints (Barták 1998).

Two methods frequently used to solve an over-constrained CSP to find the best camera position in computer games are based on constraint hierarchy and constraint weighting. In the first method, constraints are classified as hard and soft. At first, all hard constraints are satisfied. Then as many soft constraints as possible are satisfied without violating any hard constraint. In the second method, constraints are given weights and the solver calculates a trade off between constraint violations to find an optimal solution (Bourne and Sattar 2006).

As stated earlier, constraint solvers can be classified as complete or incomplete. An algorithm is complete if every solution of the problem is found by it. Incomplete

algorithms may miss some or all of the solutions. Most variants of local search solution algorithms are incomplete. In lieu of completeness, these algorithms may achieve real-time performance.

Russell and Norvig (2002) use the state space landscape shown in fig. 4 to explain local search algorithms. Search location in a landscape is determined by the current state. Cost or objective function is represented as elevation of the landscape. The aim is to find a global minimum or maximum based on whether elevation corresponds to cost or object function. A solution search may get stuck in locations such as shoulder, flat local maximum, local maximum or local minimum.

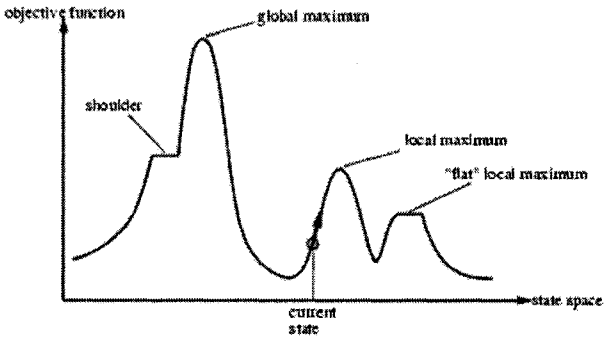

Figure 4: State Space Landscape.
(Ref: fig. 4.10; pp. 111 of Russell and Norvig 2002)

Local search CSP solving techniques use complete-state formulation. Here, every state is a complete assignment which may or may not be a solution. The next assignment is derived by changing the value of one variable at a time. Most of the local search methods use a min-conflict heuristic to choose a new value. Using the heuristic, the million-queen problem was solved in approximately 50 steps. It was also successfully applied to reduce

estimation time from three weeks to ten minutes to schedule observation of the Hubble Space Telescope for one week (Russell and Norvig 2002).

Finally, the features of an individual CSP can be exploited to arrive at solutions efficiently using specialized techniques. The structure of the CSP formulation helps to derive specific solver algorithms. The characteristics of a CSP such as number of solutions required, problem size, type of variables and constraints, structure of the constraint graph, tightness of a problem, required quality of solutions, and need of partial solutions should be considered when choosing a CSP solving technique (Tsang 1993).

2.3 Virtual Camera and Graphics Pipeline

In this section, we will discuss the placement of a virtual camera in a 3D world. We will also discuss available camera control parameters in the process of displaying 3D scene objects on a display window as 2D images. The control techniques of a virtual camera will be discussed mostly in the context of the OpenGL graphics pipeline shown in fig 5.

A graphics pipeline depicts a journey of a point from a scene of a 3D virtual world where it is modeled as a vertex to a display window. The point on the display window is defined as a pixel, i.e., picture element of a 2D image. In this process, a point undergoes four transformations, model, view, projection and viewport. All the transformations can be represented by matrices.

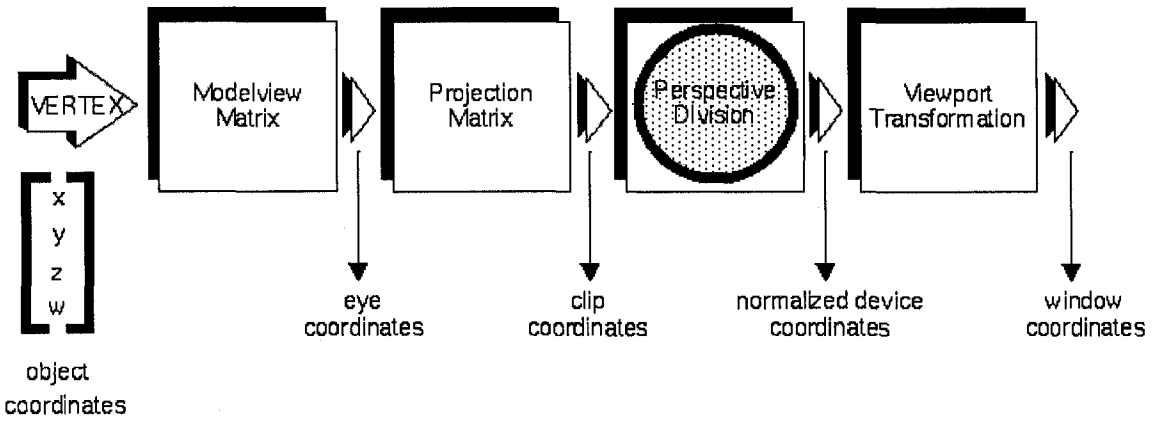

Figure 5: OpenGL Graphics Pipeline.
(Ref: fig 3-2; chapter 3 of Neider, Davis and Woo 1994)

A natural scene is composed of a set of complex objects such as a tree, forest, mountain, river, sea, fish, bird, human, building, road, bridge, and car. There is an infinite variety of objects of different size, shape, and colour. Hence, modeling a 3D scene using these complex objects is neither convenient nor intuitive. However, all complex objects can be modeled using a small set of constructs such as points, lines and polygons. These building blocks of the objects are known as output primitives (Boufama 2007). On the other hand, the basic building block of OpenGL is vertex. All the output primitives in OpenGL are defined by using vertices. A vertex can be described in a 3D homogenous coordinate system using four coordinates.

Usually, objects are defined in their own local coordinate frames. The orthogonal basis vectors of a local coordinate frame for an object are defined for easy identification and placement of parts of the object. Thus, the local coordinate frame of each object is different. Moreover, an object can be constructed using multiple parts with their own local coordinate frames. This is done especially to control independent movements of the

individual parts of an object. In this case, a part of the object can be described relative to another part of the object using appropriate transformations, and the complete object is defined by a hierarchy of transformation usually with a particular coordinate frame as the root of the tree (Verth and Bishop 2004).

2.3.1 Three Basic Transformations

The Cartesian coordinate frame in 3D consists of a standard origin (0, 0, 0) and a 3D vector space with the standard basis vectors $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$. The basis vectors are also represented by $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ where

$$\mathbf{i} = \mathbf{e}_1 = (1, 0, 0)$$

$$\mathbf{j} = \mathbf{e}_2 = (0, 1, 0)$$

$$\mathbf{k} = \mathbf{e}_3 = (0, 0, 1)$$

We can express a vector $\mathbf{v}$ of the vector space using the basis vectors as

$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}$$

The location of point P with respect to the origin of the coordinate frame will become

$$P = p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

Now, we will derive translation, rotation and scaling transformation matrices for a point defined in the coordinate frame. At first, we will consider the translation of a point shown in fig. 6. Point P is translated by a vector $\mathbf{t}$ to point Q. We have

$$\begin{aligned} Q &= P + \mathbf{t} \\ &= (p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}) + (t_x\mathbf{i} + t_y\mathbf{j} + t_z\mathbf{k}) \\ &= (p_x + t_x)\mathbf{i} + (p_y + t_y)\mathbf{j} + (p_z + t_z)\mathbf{k} \end{aligned}$$

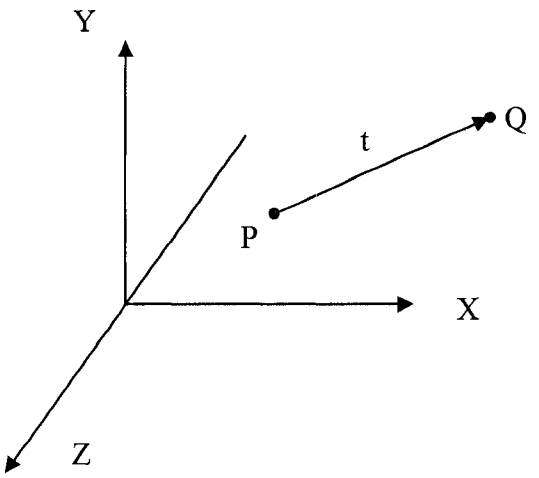

Figure 6: Translation of a Point.

Point Q can also be expressed with respect to the origin of the coordinate frame as

$$Q = q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

By equating terms of Q we have

$$q_x = p_x + t_x$$

$$q_y = p_y + t_y$$

$$q_z = p_z + t_z$$

We can write the above equation in matrix notation in homogonous coordinates as

$$\begin{pmatrix} q_x \\ q_y \\ q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

The translation transformation matrix in homogonous coordinates is

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The OpenGL function 'glTranslate()' can be used to translate a point of an object. The function creates a translation transformation matrix. Its three arguments will define the translation in their respective axes, i.e., values of t_x, t_y, and t_z.

Next, we will consider the rotation of a point. In a pure rotation, a point will be rotated in a 2D plan using the origin of the current frame as the centre of rotation. There are three such planes in 3D. Let us consider a point P in xy-plane. We will rotate P using the origin of the frame and the z-axis as the centre and the axis of rotation, respectively. Let the line between the origin and P make an angle θ with the x-axis as shown in fig. 7. P is rotated to point Q by an angle ϕ counter clockwise around the z-axis.

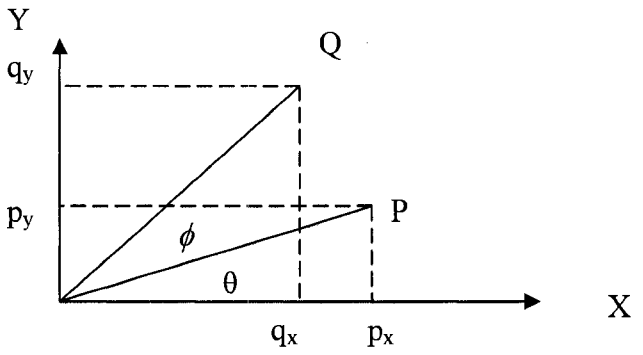

Figure 7: Rotation of a Point.

Let the distance of P and Q from the origin be r. We have

$$q_x = r \cos(\theta + \phi)$$

$$q_y = r \sin(\theta + \phi)$$

By expansion of *cos* and *sin* terms, we have

$$q_x = r \cos\theta \cos\phi - r \sin\theta \sin\phi$$

$$q_y = r \cos\theta \sin\phi + r \sin\theta \cos\phi$$

Since $r \cos\theta = p_x$, $r \sin\theta = p_y$, and $q_z = p_z$, we have

$$q_x = p_x \cos\phi - p_y \sin\phi$$

$$q_y = p_x \sin\phi + p_y \cos\phi$$

$$q_z = p_z$$

We can write the above equations in matrix notation as

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

The rotation transformation matrix around z-axis is

$$\mathbf{R}_z = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Similarly, we can derive rotation transformation matrices around y-axis and x-axis as

$$\mathbf{R}_y = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \quad \text{and} \quad \mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

Now, we can derive a general rotation transformation matrix **R** around all the three axes by applying rotation around each axis one by one as done in Verth and Bishop (2004). The angles ϕ_x, ϕ_y and ϕ_z are from transformations $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$, respectively. We have

$$\mathbf{R} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z$$

$$= \begin{pmatrix} \cos\phi_y \cos\phi_z & -\cos\phi_y \sin\phi_z & \sin\phi_y \\ \sin\phi_x \sin\phi_y \cos\phi_z + \cos\phi_x \sin\phi_z & -\sin\phi_x \sin\phi_y \sin\phi_z + \cos\phi_x \cos\phi_z & -\sin\phi_x \cos\phi_y \\ -\cos\phi_x \sin\phi_y \cos\phi_z + \sin\phi_x \sin\phi_z & \cos\phi_x \sin\phi_y \sin\phi_z + \sin\phi_x \cos\phi_z & \cos\phi_x \cos\phi_y \end{pmatrix}$$

$$= \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

We can express the rotation transformation matrix in homogonous coordinates as

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The OpenGL function 'glRotate()' can be used to rotate a point of an object. The function creates a rotation transformation matrix. It will take the degree of rotation, i.e., value of ϕ as its first argument. The remaining three arguments will define an axis of rotation.

Rotation around an arbitrary axis and a centre of rotation is also possible. The vector between the point and the centre of rotation is decomposed into component vectors that are parallel and perpendicular to the axis of rotation. The parallel component vector will remain the same during rotation. The perpendicular component vector will be rotated around the axis of rotation by the desired angle of rotation. In this case, the point will move in a plane perpendicular to the axis of rotation. Then the parallel component and the rotated perpendicular component vectors are added to find the vector between the centre of rotation and the rotated point. For mathematical derivation of rotation around an arbitrary axis and a centre of rotation, readers are directed to Verth and Bishop (2004).

Finally, let us consider scaling of a point P at (p_x, p_y, p_z). After scaling by (a, b, c) the point will become Q at (q_x, q_y, q_z). We have

$q_x = a\ p_x$

$q_y = b\ p_y$

$q_z = c\ p_z$

We can write the above equations in matrix notation as

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

The scaling transformation matrix is

$$\mathbf{S} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$$

We can express the scaling transformation matrix in homogonous coordinates as

$$\mathbf{S} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling of all points on the surface of an object will change the shape of the object. Application of the above scaling transformation with different values of a, b, and c will change the shape of the object differently in each coordinate axis. To change the shape of the object similarly in each coordinate axis, i.e., to keep the shape of the object the same, values of a, b and c should be equal. The OpenGL function 'glScale()' can be used to scale a point of an object. The function creates a scaling transformation matrix. Its three arguments will define the scale factors in their respective axes, i.e., values of a, b and c.

2.3.2 Model Transformation

The ModelView matrix in OpenGL graphics pipeline shown in fig. 5 represents two transformations, model and view. The transformations are represented by two 4×4 matrices. We derive a model transformation matrix in this sub-section. The view transformation is covered in the next sub-section.

Model transformation is a local-to-world transformation. After the model transformation, all the objects in the 3D scene will be described in a common coordinate frame known as the world coordinate frame. The OpenGL world coordinate frame is shown in fig. 8. It is a 3D coordinate frame. Thus, model transformation is a 3D to 3D transformation. The model transformation is an affine transformation.

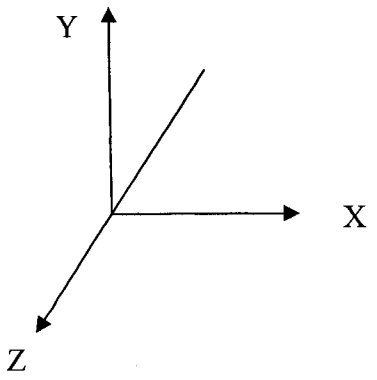

Figure 8: OpenGL World Coordination Frame.

All the objects are placed in their intended position in the 3D world using their individual local-to-world transformation. For a hierarchically defined object, the local-to-world transformation will be described as a series of transformations. A local-to-world

transformation with or without hierarchy consists of any combination of scaling, rotation and translation.

The model transformation can be described in terms of translation, rotation, and scaling transformation as

$$\mathbf{M}_{\text{local->world}} = \mathbf{TRS}.$$

2.3.3 View Transformation

View transformation is a world-to-camera transformation. After the view transformation, all the objects in the 3D scene will be described in the eye or camera coordinate frame. It is a 3D coordinate frame. Thus, view transformation is also 3D to 3D transformation. It consists of rotation and translation. It is a Euclidian transformation.

Now, we construct a view transformation matrix as done in Boufama (2007). P and Q are representing the same point defined in the world coordinate frame at (X, Y, Z) and in the camera coordinate frame at (x, y, z), respectively. The points are related by rotation and translation as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

The above equation in homogonous coordinates is

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

The view transformation matrix is

$$\mathbf{V}_{\text{world->camera}} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In OpenGL, the default camera position or eye point or viewpoint is at the origin of the world coordinate frame. The camera, by default, is oriented toward the negative z-axis of the world coordinate frame. In other words, the eye or camera coordinate frame is the same as the world coordinate frame. The objects positioned on the negative z-axis of the eye or camera coordinate frame will be displayed at the centre of the display window or viewport.

Camera position and camera orientation are two main camera control parameters. These parameters can be changed arbitrarily anytime using rotation and translation. In OpenGL, we can again use the 'glTranslate()' and 'glRotate()' functions to change the camera position and the camera orientation. Changing the parameters is equivalent to moving and orienting the camera physically in the real world. It defines a new eye or camera coordinate frame. The new camera position defines the origin of the new frame. The negative z-axis of the new frame will be aligned with the camera optical axis.

The 'gluLookAt()' function, defined in the OpenGL utility library, is a more convenient function to change the current camera position and to define a new line of sight (Neider, Davis and Woo 1994). The function will take a new camera position, a look at object

position, and an up vector as arguments. With the function call, a new camera coordinate frame consists of normalized orthogonal basis vectors will be defined from the three arguments. The origin of the new frame will be at the new camera position. The camera will be oriented toward the negative z-axis of the new frame which is aligned with the look at vector between the camera position and the look at object position. Because the objects positioned on the negative z-axis of the camera coordinate frame will be displayed at the centre of the display window, any object can be displayed at the centre or at any position relative to the centre of the display window by changing the 'look at' direction of the camera.

Now, we will derive three orthonormal basis vectors **u**, **v** and **n** representing x-axis, y-axis and z-axis of an eye coordinate frame respectively from a look at vector and an up vector as done in Drucker (1994). Let the look at vector be $\mathbf{V}_{\mathbf{lookAt}}$, the up vector be $\mathbf{V}_{\mathbf{up}}$, and **n** be the negative of the normalized vector of $\mathbf{V}_{\mathbf{lookAt}}$. In other words, the look at vector $\mathbf{V}_{\mathbf{lookAt}}$ will align with the negative z-axis of the eye coordinate frame as in OpenGL. Let the projection of $\mathbf{V}_{\mathbf{up}}$ on **n** be $\mathbf{V}_{||}$ and the component of $\mathbf{V}_{\mathbf{up}}$ perpendicular to **n** be $\mathbf{V}_{\mathbf{L}}$. We have

$$\mathbf{V}_{||} = \frac{\mathbf{V}_{up} \bullet \mathbf{n}}{|\mathbf{n}|^2}\mathbf{n} = (\mathbf{V}_{up} \bullet \mathbf{n})\mathbf{n}$$

$$\mathbf{V}_{L} = \mathbf{V}_{up} - \mathbf{V}_{||}$$

$$\mathbf{v} = \frac{\mathbf{V}_{L}}{|\mathbf{V}_{L}|}$$

$$\mathbf{u} = \mathbf{v} \times \mathbf{n}$$

The Gram-Schmidt Orthogonalization is another method of defining orthonormal basis vectors from non-orthonormal basis vectors (Verth and Bishop 2004).

We can also calculate an eye or camera coordinate frame from a camera position and three Euler angles. The Euler angles are pitch, yaw and roll. They are also known as azimuth, elevation and roll. Pitch, yaw and roll are rotations around x-axis, y-axis, and z-axis respectively.

In the first person perspective camera, the camera position is fixed relative to the player character and the look at vector is along the line between the camera and the object of interest. In the second person perspective camera, the camera position is fixed relative to an antagonist of the player character and the look at vector is along the line between the camera and the player character. In the third person perspective camera, the camera position can be at any arbitrary point in the 3D world and the look at vector is along the line between the camera position and the objects of interest.

We can implement basic cinematic camera movement by rotation and translation. The tilt, pan and roll camera movements can be achieved by rotation of the camera around x-axis, y-axis and z-axis of the world coordinate frame, respectively. On the other hand, the dolly, crane and truck camera movement can be achieved by the translation of the camera position along x-axis, y-axis and z-axis of the world coordinate frame, respectively.

By changing the arguments of the 'gluLookAt()' function, we can place the camera in any of the positions or orientations shown in fig. 9 to implement various cinematic shots. We discuss fully the various methods of camera placement at the desired position and orientation by the constraint and non-constraint based camera control systems in the next two sections.

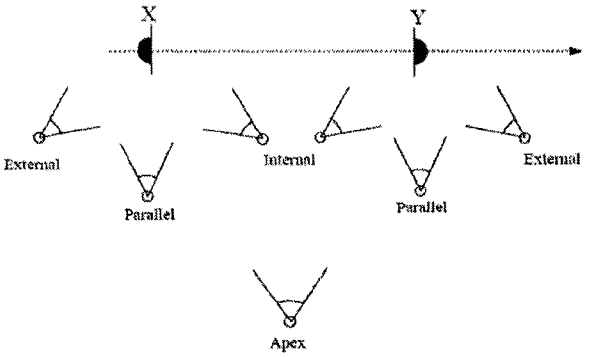

Figure 9: Camera Placement Relative to the Line of Interest.
(Ref: adapted from fig. 1 of Christianson et al. 1996 which is adapted from fig. 4.11 of Arijon 1976)

2.3.4 Projection Transformation

Before the projection, culling of the scene is done. The objects which lie completely outside the view frustum are ignored for the rest of the transformations for the shot. A view frustum is defined by six planes. The planes are determined by the type of projection. The view frustum for a parallel projection has two pairs of parallel planes defined by opposite sides of the view window, in addition to near and far view planes. On the other hand, four planes of the view frustum for a perspective projection are defined by two adjacent vertices of the view window and the view point. The remaining two planes are near and far view planes. A view frustum or viewing volume for a perspective projection is shown in fig. 10. An efficient culling method can use object bounding boxes

in different resolutions to test if an object is completely outside the viewing volume (Verth and Bishop 2004).

After culling, the 3D objects inside the view frustum are projected to a 2D coordinate frame using the Projection Matrix shown in fig. 5. Thus, this transformation is 3D to 2D. The projection transformation is represented by a 3×4 matrix. It is a projective transformation. There are four types of projection, namely, oblique-parallel, orthographic-parallel, oblique-perspective, and perspective projections.

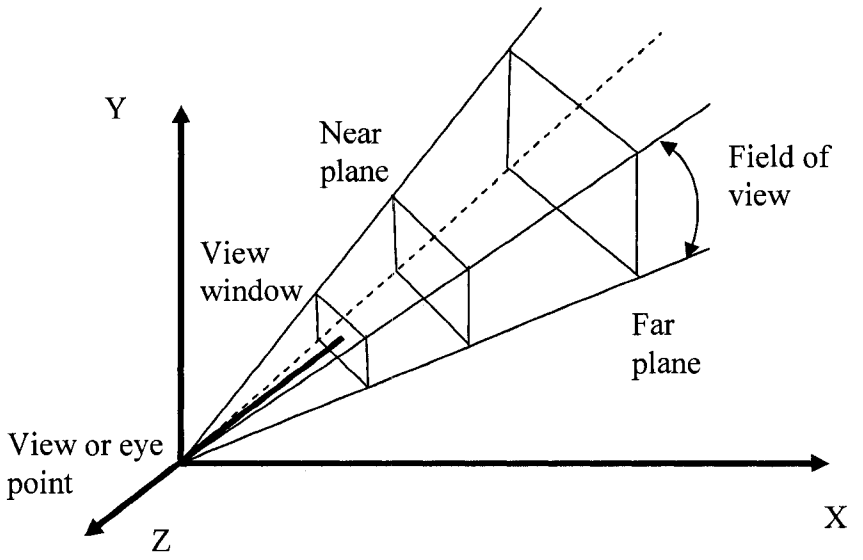

Figure 10: View Frustum.

In parallel projection, the size of an object will remain the same irrespective of the distance of the object from the view point. The centre of the projection is assumed at infinity. The fixed size of an object creates odd views if used for a 3D scene such as in computer games. But, keeping parallel lines parallel in the displayed image is important for accurate measurement in blueprints and construction drawings. Thus, parallel

projection is used mainly in CAD/CAM (Verth and Bishop 2004). In OpenGL, we can use the 'glOrtho()' function to implement orthographic-parallel projection.

Foreshortening is the characteristics of the perspective projection where an object appears smaller as it retreats from the camera (Neider, Davis and Woo 1994). The perspective projection is similar to our perceived view of the real world (Verth and Bishop 2004), which is why perspective projections are mainly used in computer games to simulate real world views. The working principle of perspective projection is similar to the working principle of a camera in the physical world. However, to avoid inverted images of the objects, the view plane is placed in front of the view point or centre of projection.

Now, we will construct a projection transformation matrix mostly based on the work of Verth and Bishop (2004) without considering culling and clipping. For derivation of a projection transformation with culling and clipping, readers are directed to Verth and Bishop (2004).

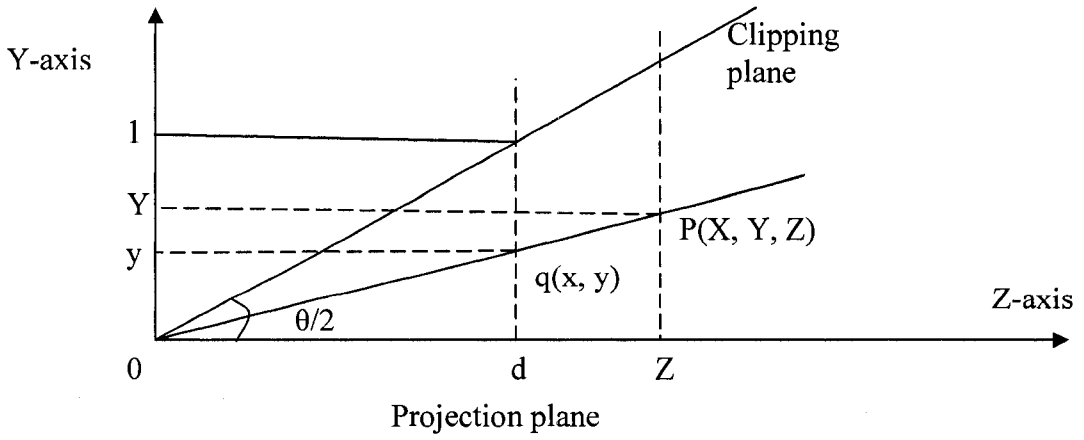

Figure 11: Perspective Projection.

In perspective projection, a 3D point P(X, Y, Z) is projected to the 2D point q(x, y) on the view or projection plane having an aspect ratio a at a distance d from the view point. The point q will be expressed in NDC frame having dimensions of all the axes in the [-1, 1] range.

Let us consider projection of P to q in yz-plane as shown in fig. 11. From the similar triangles we have

$$\frac{Y}{y} = \frac{Z}{d}$$

Similarly, considering projection in xz-plane and from the similar triangles we have

$$\frac{X}{ax} = \frac{Z}{d}$$

By rearranging the above equations we have

$$x = \frac{d}{a}\frac{X}{Z}$$

$$y = d\frac{Y}{Z}$$

Putting $\lambda = 1/Z$, the above equations can be written in homogenous coordinates in matrix notation as

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} \frac{d}{a} & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

All the points on the line from the view point or centre of projection through the point P will be projected to the same point q on the projection plane. This fact is implied by the

presence of the scale factor λ in the above equation. In other words, λ is a scale for equality in the projective space. We have projection transformation matrix as

$$\mathbf{P}_{\text{camera->NDC}} = \begin{pmatrix} \frac{d}{a} & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

In OpenGL, 'glFrustum()' function can be used for projection. The function creates a view frustum by defining the lower-left and upper-right corners of the near clipping plane. Thus, four planes are defined by the adjacent corners of the near clipping plane and the view point. The two remaining planes are the near clipping and the far clipping planes. The far clipping plane is parallel to the near clipping plane and is at the specified distance from the view point. This function can be used to implement both perspective and oblique-perspective projections. The oblique-perspective projection can provide dramatic and special effects of visualization.

Alternatively, we can use the OpenGL utility library function 'gluPerspective()' to implement perspective projection. The function creates a symmetric perspective view frustum. It is easy and intuitive to control the view frustum by calling the function. It takes the angle between the upper and lower clipping planes known as FOV as its first argument. The function also takes the ratio of width to height of the near plane of the view frustum called aspect ratio, and the distances of the near and far clipping planes from the view point as arguments.

Now, let us find an expression relating FOV θ and the distance d between the centre of the projection and the projection plane as shown in fig. 11. The angle between the negative z-axis and the upper clipping plane will be $\frac{\theta}{2}$. The distance between the negative z-axis and the top most projected point in the Normalized Device Coordinate (NDC) frame is 1. We have

$$\tan\frac{\theta}{2} = \frac{1}{d}$$

$$d = \frac{1}{\tan\frac{\theta}{2}} = \cot\frac{\theta}{2}$$

If we keep the height of the view plane constant as in NDC, any change of the FOV angle will change the distance between the view plane and the view point. By changing the FOV angle to large and small values, we can implement wide-angle and telephoto lenses, respectively. Thus, different cinematic shot types, shown in fig. 12, can be implemented by changing the FOV angle along with different camera positions and orientations.

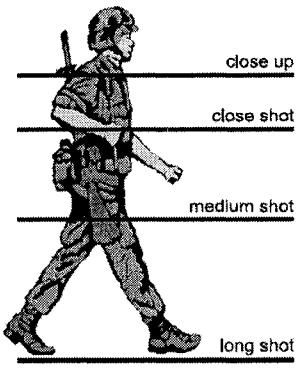

Figure 12: Different Shot Types with their Cutoff Positions.
(Ref: fig. 2.4; pp 36 of Bourne 2006)

After projection transformation, clipping is done. All the outside portions of the edges of the objects, which are partially inside the view volume, are clipped. During clipping, OpenGL reconstructs the clipped polygon edges (Neider, Davis and Woo 1994).

In the Perspective Division phase of the OpenGL graphic pipeline, coordinate values of the vertices are divided by 'w' to derive the coordinates of the vertices in the NDC frame (Neider, Davis and Woo 1994).

2.3.5 Viewport Transformation

Now, we construct a viewport transformation matrix as done in Boufama (2007). The viewport transformation makes a correspondence between the points transformed into the NDC frame and the screen image pixels in the window coordinate frame (Neider, Davis and Woo 1994), creating a 2D to 2D transformation. The viewport transformation is represented by a 3×3 matrix. It is an affine transformation.

Let p(x, y, 1) and q(u, v, 1) represent the same point defined in a scene unit in the NDC frame and in pixels in the window or viewport coordinate frame, respectively. In OpenGL, the scene unit can be imagined in any unit such as ft, miles, mm, cm, m, and km. Let us consider that the scene unit is given in mm. Now, we have to change the unit of the coordinate values from mm to pixels using scaling. The scaling factors α_u and α_v can be defined in terms of focal length in mm (f), the number of pixels per mm along the x-axis (K_u), and the number of pixels per mm along the y-axis (K_v) as

$$\alpha_u = fK_u$$

$$\alpha_v = fK_v$$

In addition, we have to consider the change of the origins of the frames. The origin of NDC is located at the point of intersection of the camera optical axis with the view plane. The origin of NDC will be mapped into the centre of the viewport. The origin of the window or viewport coordinate frame is at the upper-left corner of the viewport. The origin of NDC is located at the point (u_0, v_0) in the window or viewport coordinate frame where both u_0 and v_0 are in pixels. Thus, the coordinates of point p and q are related by the equations

$$u = \alpha_u x + u_0$$
$$v = \alpha_v y + v_0$$

We can write the above equations in matrix notation in homogonous coordinates as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The viewport transformation matrix is

$$\mathbf{A}_{\text{NDC->viewport}} = \begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

In openGL, viewport transformation is performed by calling the 'glViewport()' function. The function uses x and y coordinates of a corner as well as the width and height of the viewing window. To get a distortion-free projected image, the aspect ratio of the view port should be equal to the aspect ratio of the view volume as defined in the projection phase of the graphics pipeline (Neider, Davis and Woo 1994).

2.3.6 Overall Transformation Matrix and its Uses

Let us summarize the process of object transformation. At the beginning, all 3D scene objects will be in their own local coordinate frames. They will be placed in the world coordinate frame by the model transformation. The camera will also be placed in the world coordinate frame by defining an eye coordinate frame relative to the world coordinate frame. In fig. 13, vector **t** represents distance and relative orientation between the world and the eye coordinate frames. After the view transformation, all 3D scene objects will be in the eye coordinate frame. After the projection transformation, all 3D scene objects will become 2D objects in the NDC frame. Finally, the objects will become 2D images after the viewport transformation. The image will be in the window coordinate frame. The relationship between the various coordinate frames is shown in fig. 13.

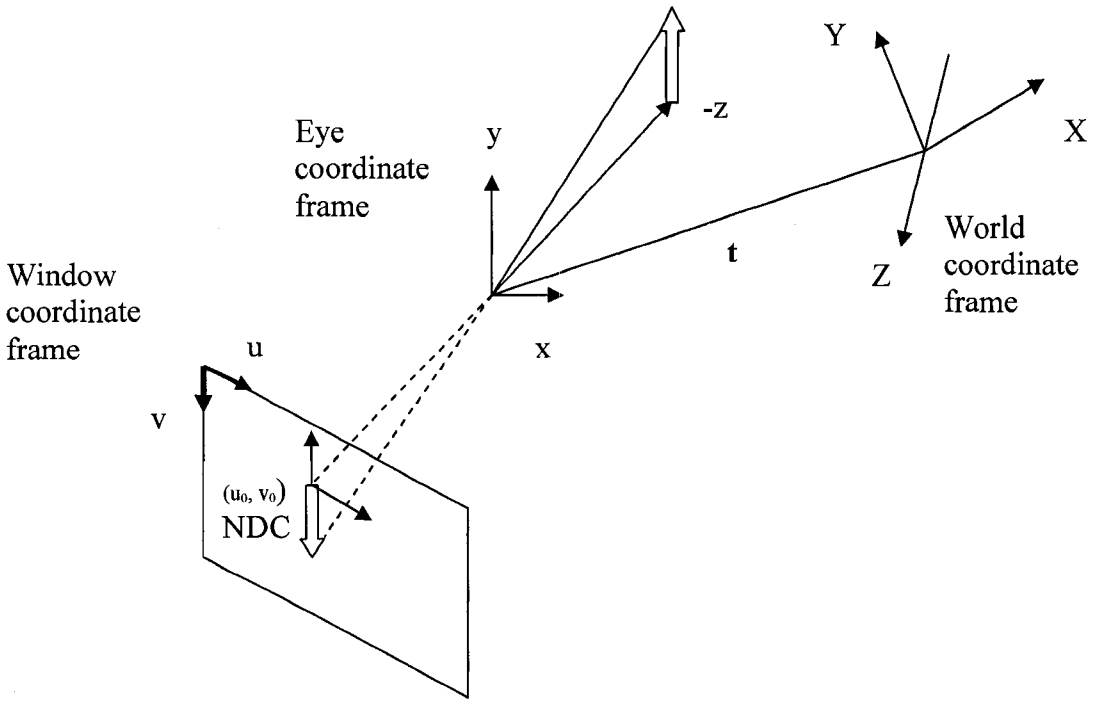

Figure 13: Relation between the Coordinate Frames.
(Adapted from fig. 1; pp. 4; chap. 3 of the slides for 60-551 of Boufama 2007)

Now, let us consider the application of model, view, projection and viewport transformation to a point P(X, Y, Z, 1) defined in its local coordinate frame, in that order. The point P(X, Y, Z, 1) will be converted to the image point q(u, v, 1) defined in pixels in the window coordinate frame by the transformations. We have

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \, \mathbf{A}_{\text{NDC->viewport}} \, \mathbf{P}_{\text{camera->NDC}} \, \mathbf{V}_{\text{world->camera}} \, \mathbf{M}_{\text{local->world}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Putting $\mathbf{M} = \mathbf{A}_{\text{NDC->viewport}} \, \mathbf{P}_{\text{camera->NDC}} \, \mathbf{V}_{\text{world->camera}} \, \mathbf{M}_{\text{local->world}}$, we have

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \, \mathbf{M} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$q = \lambda \, \mathbf{M}P$$

M is a 3×4 matrix representing four transformations of a point from 3D to 2D. In the physical world, camera calibration is equivalent to finding the elements of **M**. **M** has 12 elements, 11 of which are independent.

Christie and Olivier (2006) derived another camera model and over-all transformation matrix consisting of translation of camera position, rotation on the Euler angles, and projection. They observed that the relationship between the 3D and 2D coordinates of a point is highly non-linear. Thus, it is difficult to invert the over-all transformation matrix. In other words, it is difficult to find the 3D coordinates of a point from its known 2D coordinates by inversion of the over-all transformation matrix. Fortunately, there are only

a few applications such as picking that require 3D coordinates of a point from its 2D coordinates.

We can define an image declaratively by constraining the 2D image coordinates of objects as done in many camera control systems such as in Jardillier and Languénou (1998). In a 3D environment, the 3D coordinates of the static objects are known and the 3D coordinates of the dynamic objects are almost always known. If we can derive a transformation matrix **M** from known 3D and 2D coordinates, the 2D image constraints can be satisfied by using it for projection.

Now, we will outline a solution technique to calculate the elements of **M** from known 3D and 2D coordinates as done in Boufama (2007). By using the elements of matrix **M**, we can write the equation $q = \lambda \mathbf{MP}$ as

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

The values of u and v can be calculated as

$$u = \frac{m_{11}X + m_{12}Y + m_{13}Z + m_{14}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

$$v = \frac{m_{21}X + m_{22}Y + m_{23}Z + m_{24}}{m_{31}X + m_{32}Y + m_{33}Z + m_{34}}$$

We have the following linear equations by rearranging the above equations

$$m_{11}X + m_{12}Y + m_{13}Z + m_{14} - um_{31}X - um_{32}Y - um_{33}Z - um_{34} = 0$$
$$m_{21}X + m_{22}Y + m_{23}Z + m_{24} - vm_{31}X - vm_{32}Y - vm_{33}Z - vm_{34} = 0$$

Assuming both the 2D image coordinates and the 3D scene coordinates of a point are known, the 3D coordinates of a point P_i and the corresponding 2D coordinates of the point q_i provide us with two equations. For n points, we have a set of $2n$ equations. We need at least six points defined both in 3D and 2D to find the values of the elements of **M**. We can write the linear equations in matrix notation as

$$\begin{pmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1Z_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -v_1X_1 & -v_1Y_1 & -v_1Z_1 & -v_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -u_2X_2 & -u_2Y_2 & -u_2Z_2 & -u_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -v_2X_2 & -v_2Y_2 & -v_2Z_2 & -v_2 \\ . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -u_nX_n & -u_nY_n & -u_nZ_n & -u_n \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -v_nX_n & -v_nY_n & -v_nZ_n & -v_n \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix} = 0$$

The above equation becomes **AV** = 0. Numerical methods such as the singular value decomposition (SVD) technique can be used to solve the above equation.

Once elements m_{11}, m_{12}, m_{13}, ..., m_{34} of **V** are calculated, a 3×4 matrix **M** can be constructed easily, allowing **M** to be used to project the 3D objects to the desired location in a 2D image.

2.4 Camera Control Systems

The desired camera position can be described mathematically. Let a target be at the origin of a Cartesian coordinate frame. The desired camera position can be described relative to the target in a 3D spherical coordinate as done in Stone (2004) using a vector (ρ, θ, ϕ), where ρ is the distance to the origin, θ is the counter-clockwise rotation angle around the y-axis starting from the z-axis, and ϕ is the signed rotation angle from xz-plane. A camera position in a spherical coordinate is shown in Fig. 14. The corresponding coordinates of the camera position in a Cartesian frame are

$$x = \rho \cos \phi \sin \theta$$

$$y = \rho \sin \phi$$

$$z = \rho \cos \phi \cos \theta$$

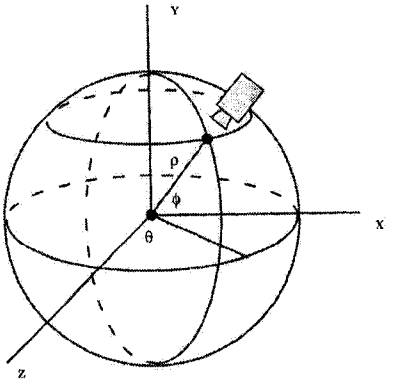

Figure 14: Camera Position in Spherical Coordinate.
(Adapted from fig. 4.1.1; pp 304 of Stone 2004)

In a dynamic environment, the target position is not fixed at any single point. If we add the coordinates of the current target position to the initial camera coordinates, i.e., x, y,

and z calculated above, we will get a new camera position. The new camera position will maintain displacement defined by the vector (ρ, θ, ϕ) relative to the target.

Variation to the camera control scheme can be implemented easily. The angle θ can be a counter-clockwise rotation angle around the y-axis starting from the target facing. The desired relative camera height of a camera can be used instead of ϕ (Bourne 2006).

Direct mathematical methods are the simplest ways to describe a desired camera position. Direct mathematical methods depend on other methods such as spring systems to handle acceleration and deceleration during camera movements (Treglia 2000).

Blinn (1988) uses vector algebra to define camera positions to make space movies. Given the image coordinates of a spacecraft (x_f, y_f) and a planet (x_a, y_a), as well as the distance between the spacecraft and the eye point d in world space, the camera control problem is defined so as to find the displacement vector **D** of the eye point from the spacecraft and the look at vector **T**. The world space and the corresponding view space are shown in fig. 15a and fig. 15b respectively.

Now, we will drive equations of **T** and **D** as outlined in Blinn (1988). All the vectors used in this derivation are row vectors. Let the FOV angle be ϕ. The view plan will be at $-\cot\frac{\phi}{2}$ from the eye point. Thus, z-coordinates of all the points on the view plane are $-\cot\frac{\phi}{2}$.

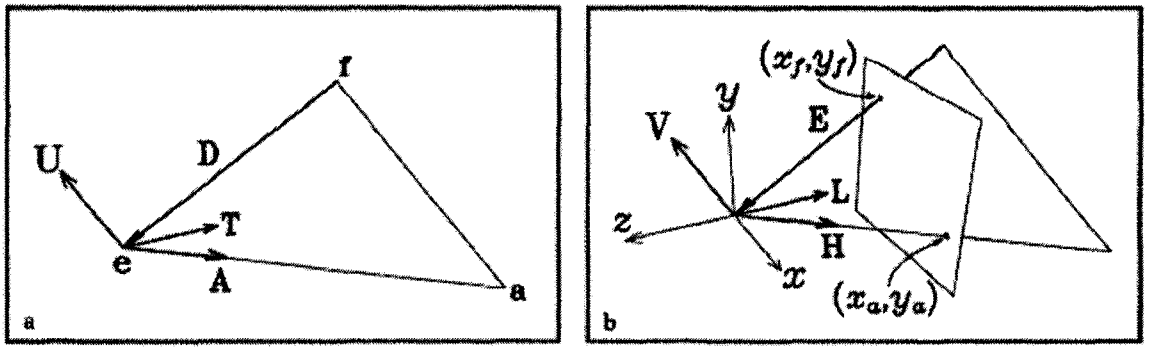

Figure 15: The World and the View Spaces.
(Ref: fig. 1; pp. 78 of Blinn 1988)

At the beginning, we will derive equations of the displacement vector **E** of the eye point from the spacecraft and a normalized vector **H** showing direction from eye point to the planet. Vectors **E** and **H** are in view space. The vector from the eye point to the spacecraft location point on the view plane is $(x_f, y_f, -\cot\frac{\phi}{2})$. We will normalize the vector and multiply by $-d$ to find the value of **E**. The vector from the eye point to the planet location point on the view plane is $(x_a, y_a, -\cot\frac{\phi}{2})$. We will normalize the vector to find the value of **H**. We have

$$\mathbf{E} = \left(\frac{-d\,x_f}{\sqrt{x_f^2 + y_f^2 + \cot^2\frac{\phi}{2}}}, \frac{-d\,y_f}{\sqrt{x_f^2 + y_f^2 + \cot^2\frac{\phi}{2}}}, \frac{d\cot\frac{\phi}{2}}{\sqrt{x_f^2 + y_f^2 + \cot^2\frac{\phi}{2}}} \right)$$

$$\mathbf{H} = \left(\frac{x_a}{\sqrt{x_a^2 + y_a^2 + \cot^2\frac{\phi}{2}}}, \frac{y_a}{\sqrt{x_a^2 + y_a^2 + \cot^2\frac{\phi}{2}}}, \frac{-\cot\frac{\phi}{2}}{\sqrt{x_a^2 + y_a^2 + \cot^2\frac{\phi}{2}}} \right)$$

We then derive the equation of the vector from the spacecraft to the planet **R** in view space. The corresponding vector from the spacecraft to the planet **Q** in world space is known. We have $|\mathbf{E}| = d$ and $|\mathbf{H}| = 1$. From fig. 16b we have

$$-\mathbf{E}.\mathbf{H} = |-\mathbf{E}||\mathbf{H}| \cos\theta = |\mathbf{E}||\mathbf{H}| \cos\theta = d \cos\theta$$

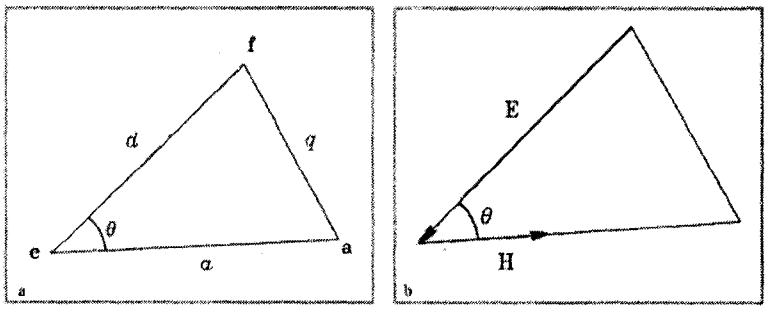

Figure 16: Triangles in the World and the View Spaces.
(Ref: fig. 2; pp. 81 of Blinn 1988)

Using law of cosine in the world space triangle in fig. 16a,

$$q^2 = a^2 + d^2 - 2\,a\,d\cos\theta$$

$$a^2 - 2\,d\cos\theta\,a + d^2 - q^2 = 0$$

$$a = d\cos\theta \pm \sqrt{(d\cos\theta)^2 - d^2 + q^2}$$

Putting value of $d \cos\theta$,

$$a = -(\mathbf{E}.\mathbf{H}) \pm \sqrt{(\mathbf{E}.\mathbf{H})^2 - d^2 + q^2}$$

Thus, we have

$$\mathbf{R} = \mathbf{E} + a\mathbf{H}$$

Now, we will derive the equation for the up vector **V** in view space. The corresponding up vector **U** in world space is known. Both are unit vectors. The x-coordinate of **V** is zero

i.e. $V_x = 0$. We need to calculate two more coordinates of **V**, i.e., V_y and V_z, only. From the dot product constancy across the coordinate frames, we have

$$\mathbf{V.R} = \mathbf{U.Q}$$

The equation in terms of axes coordinates becomes

$$V_y R_y + V_z R_z = \mathbf{U.Q}$$

$$V_z R_z = \mathbf{U.Q} - V_y R_y$$

From the dot product of a unit vector, **V.V** = 1. The equation in terms of axes coordinates becomes

$$V_y^2 + V_z^2 = 1$$

Multiplying both sides of the equation by R_z^2 and putting the value of $V_z R_z$, we have

$$V_y^2 R_z^2 + (\mathbf{U.Q} - V_y R_y)^2 = R_z^2$$

$$(R_y^2 + R_z^2)\, V_y^2 - 2(\mathbf{U.Q})\, R_y\, V_y + (\mathbf{U.Q})^2 - R_z^2 = 0$$

By solving the quadratic equation, we have

$$V_y = \frac{(\mathbf{U.Q})R_y \pm R_z\sqrt{R_y^2 + R_z^2 - (\mathbf{U.Q})^2}}{R_y^2 + R_z^2}$$

Similarly,

$$V_z = \frac{(\mathbf{U.Q})R_z \mp R_y\sqrt{R_y^2 + R_z^2 - (\mathbf{U.Q})^2}}{R_y^2 + R_z^2}$$

Now, we will derive the equation for the look at vector **T** in world space. To calculate **T**, we can express **T** as a linear combination of **Q**, **U**, and a cross product of **Q** and **U**. Then we have to find the values of α, β, and γ.

$$\mathbf{T} = \alpha\mathbf{Q} + \beta\mathbf{U} + \gamma(\mathbf{Q}\times\mathbf{U})$$

Before finding the values of α, β, and γ, let us simplify $(\mathbf{Q}\times\mathbf{U}).(\mathbf{Q}\times\mathbf{U})$ using vector identities. Assuming $\mathbf{Q}\times\mathbf{U} = \mathbf{W}$ is the first vector of the dot product, we have

$$(\mathbf{Q}\times\mathbf{U}).(\mathbf{Q}\times\mathbf{U}) = \mathbf{W}.(\mathbf{Q}\times\mathbf{U}) = \mathbf{Q}.(\mathbf{U}\times\mathbf{W}) = \mathbf{U}.(\mathbf{W}\times\mathbf{Q})$$

$$= \mathbf{U}.[(\mathbf{Q}\times\mathbf{U})\times\mathbf{Q}] \quad \text{putting } \mathbf{W} = \mathbf{Q}\times\mathbf{U} \text{ back}$$

$$= \mathbf{U}.[(\mathbf{Q}.\mathbf{Q})\mathbf{U} - (\mathbf{U}.\mathbf{Q})\mathbf{Q})] = (\mathbf{Q}.\mathbf{Q})\,(\mathbf{U}.\mathbf{U}) - (\mathbf{U}.\mathbf{Q})^2$$

$$= (\mathbf{Q}.\mathbf{Q}) - (\mathbf{U}.\mathbf{Q})^2$$

Assuming $\Delta = (\mathbf{Q}.\mathbf{Q}) - (\mathbf{U}.\mathbf{Q})^2$, we have

$$(\mathbf{Q}\times\mathbf{U}).(\mathbf{Q}\times\mathbf{U}) = \Delta$$

Now, let us calculate the values of α, β, and γ. The view direction vector $\mathbf{L}$ in view space is known.

$$\mathbf{L} = (0, 0, -1)$$

From the dot product constancy across the coordinate frames, we have

$$\mathbf{V}.\mathbf{L} = \mathbf{U}.\mathbf{T}$$

Putting $\mathbf{U}.\mathbf{U} = 1$ and the values of $\mathbf{V}$, $\mathbf{L}$, and $\mathbf{T}$ in the above equation, we have, in terms of axes coordinates,

$$-V_z = \alpha\,(\mathbf{U}.\mathbf{Q}) + \beta$$

$$\beta = -V_z - \alpha\,(\mathbf{U}.\mathbf{Q})$$

Again, from the dot product constancy across the coordinate frames, we have

$$\mathbf{R}.\mathbf{L} = \mathbf{Q}.\mathbf{T}$$

Putting the values of $\mathbf{R}$, $\mathbf{L}$, and $\mathbf{T}$ in the above equation, we have, in terms of axes coordinates,

$$-R_z = \alpha\,(\mathbf{Q}.\mathbf{Q}) + \beta\,(\mathbf{U}.\mathbf{Q})$$

Putting the value of β in the above equation, we have

$$-R_z = \alpha\,(\mathbf{Q}.\mathbf{Q}) - V_z\,(\mathbf{U}.\mathbf{Q}) - \alpha\,(\mathbf{U}.\mathbf{Q})^2$$

$$\alpha\,[\mathbf{Q}.\mathbf{Q} - (\mathbf{U}.\mathbf{Q})^2] = -R_z + V_z\,(\mathbf{U}.\mathbf{Q})$$

Since $\Delta = \mathbf{Q}.\mathbf{Q} - (\mathbf{U}.\mathbf{Q})^2$, we have

$$\alpha = [-R_z + V_z\,(\mathbf{U}.\mathbf{Q})] / \Delta$$

Putting the value of α in the equation of β and simplifying the equation, we have

$$\beta = [-V_z\,(\mathbf{Q}.\mathbf{Q}) + R_z\,(\mathbf{U}.\mathbf{Q})] / \Delta$$

Again, from the dot product constancy across the coordinate frames, we have

$$(\mathbf{R}\times\mathbf{V}).\mathbf{L} = (\mathbf{Q}\times\mathbf{U}).\mathbf{T}$$

Putting the values of **L, T**, and $(\mathbf{Q}\times\mathbf{U}).(\mathbf{Q}\times\mathbf{U})$ in the above equation, we have, in terms of axes coordinates,

$$-R_x V_y = \gamma(\mathbf{Q}\times\mathbf{U}).(\mathbf{Q}\times\mathbf{U})$$

$$\gamma = -R_x V_y / \Delta$$

Now, we will derive the 3×3 transformation matrix **M** using **T** and **U**. Let $\mathbf{M} = (\mathbf{M}_{i1}, \mathbf{M}_{i2}, \mathbf{M}_{i3})$. Since **M** represents pure rotation, the inverse of **M** is equal to the transpose of **M**. It transforms **T** and **U** in world space to **L** and **V** in view space, respectively.

$$\mathbf{TM} = \mathbf{L}$$

$$\mathbf{UM} = \mathbf{V}$$

Post multiplying both sides of the equations by $\mathbf{M}^T$, we have

$$\mathbf{T} = \mathbf{L}\mathbf{M}^T$$

$$\mathbf{U} = \mathbf{V}\mathbf{M}^T$$

Putting value of **L** in the equation of **T**, we have

$$\mathbf{T} = (0, 0, -1)\mathbf{M}^T = -\mathbf{M}_{i3}$$

$$\mathbf{M}_{i3} = -\mathbf{T}$$

Putting the value of **V** in the equation of **U**, we have

$$\mathbf{U} = (0, V_y, V_z)\mathbf{M}^T = V_y \mathbf{M}_{i2} + V_z \mathbf{M}_{i3}$$

Putting the values of $\mathbf{M}_{i3}$ and rearranging terms, we have

$$\mathbf{M}_{i2} = \frac{\mathbf{U} + V_z \mathbf{T}}{V_y}$$

From the dot product constancy across the coordinate frames, we have

$$\mathbf{L}.\mathbf{V} = \mathbf{T}.\mathbf{U}$$

Putting the values of **L** and **V** in the above equation, we have, in terms of axes coordinates,

$$(0, 0, -1).(0, V_y, V_z) = \mathbf{T}.\mathbf{U}$$

$$V_z = -\mathbf{T}.\mathbf{U} = -\mathbf{U}.\mathbf{T}$$

Putting values of V_z in the equation of $\mathbf{M}_{i2}$, we have

$$\mathbf{M}_{i2} = \frac{\mathbf{U} - (\mathbf{U}.\mathbf{T})\mathbf{T}}{V_y}$$

From the cross product of $\mathbf{M}_{i2}$ and $\mathbf{M}_{i3}$, we have

$$\mathbf{M}_{i1} = \mathbf{M}_{i2} \times \mathbf{M}_{i3} = -\frac{\mathbf{U} \times \mathbf{T}}{V_y}$$

Finally, we will calculate **D**. We can transform **D** to **E** by using **M.**

$$\mathbf{D}\mathbf{M} = \mathbf{E}$$

Post multiplying both the sides of the equation by $\mathbf{M}^T$, we have

$$\mathbf{D} = \mathbf{E}\mathbf{M}^T$$

Wane and Osborne (1990) propose three metaphors for camera control and exploration of the virtual environment using input devices having six DOFs. The metaphors are 'eyeball in hand', 'scene at hand', and 'flying vehicle control'. In the 'eyeball in hand' metaphor, the view point can be moved around in a 3D world. A 2D image contains only those objects which are visible from the view point. In the 'scene at hand' metaphor, translation and rotation of the control device will generate the corresponding translation and rotation of the 3D world. The 'flying vehicle control' metaphor allows the user to fly the view point through the 3D world. The camera control system using metaphors is one of the early high-level camera control systems in the 3D environment.

He, Cohen and Salesin (1996) propose a camera control system that consists of a real-time application, a camera control system called Virtual Cinematographer (VC), and a renderer. The components of the VC are grouped into two sets: idioms and camera modules. Idioms represent cinematographic knowledge and expertise. Each idiom will capture a particular type of scene by selecting the appropriate shots and by controlling transition timing between shots. An idiom is represented as a finite state machine. Finite state machines are organized hierarchically with more generic idioms at higher levels. Thus, in the case of any unforeseen events, an idiom will return control to a more general idiom. A camera module will handle camera placement geometrically based on the type of shot and number of actors. It can also make minor changes to actor positions for better effect. A camera module can be utilized by any idiom. Sixteen camera modules are implemented. At each time tick, the game application will send the VC a description of events which are significant to the protagonist. The VC uses the events along with the

existing state of animation at that point of time to produce a camera specification for the time tick, sending the specification to the renderer. At the same time tick, the game application supplies the renderer with animation parameters and a description of the current application. To display game contents, the renderer uses the information from both the game application and the VC for the same time tick. The system is tested with an interactive game called 'virtual party', played by multiple players over a network and in real-time performance.

Halper and Olivier (2000) propose a camera planning agent called CAMPLAN for camera placement to capture a shot with some predefined set of visual properties. Image properties of an object in the scene can be defined by its position, size, visibility or orientation with respect to viewplane, viewport and viewpoint of the camera, either relatively or absolutely. A set of functions to define image properties declaratively are classified and implemented. Once a user of the system selects the shot properties, COMPLAN utilizes a genetic algorithm to find a camera placement that maximizes the selected shot properties.

Hornung, Lakemeyer and Trogemann (2003) formalize the dramaturgical means of expression of the movie camera system to be applied in interactive narratives, and identify a set of basic dramaturgical principles. The formalization is implemented in a camera agent. In the camera agent, a neural network called perceptron is used to choose the appropriate camera shots for a given situation within the current narrative context. A story is represented within the camera agent as narrative events in such a way that a

narrative event corresponds to a single story event. The narrative application sends information regarding its states as narrative events to the camera agent. The system uses eight parameters of events to communicate information about the states of a story between the narrative application and the camera agent. After receiving an event, the camera agent re-computes its internal representation of the narrative based on the events received so far. Then, the camera agent selects an event as the active one for visualization based on the history of the narrative, coherence between the events, and event priority. Next, the decision module of the camera agent selects matching shots from a user-defined shot library to be added to a priority list of the active event. Finally, the action-realization module of the camera agent selects a shot with the highest priority for the active event. The module computes camera position and orientation per cinematic rule for the shot, and transfers the information back to the narrative application for visualization. If a shot can not be realized, the next matching shot will be considered for the active event.

Giors (2004) describes the camera control system of Full Spectrum Warrior (FSW). FSW has several cameras. Each camera is derived from an abstract base class and is managed by a camera manager. The main camera represents the primary user controlled view and the fly-by system. Other cameras are used for cinematic, in-game events and playback. The cinematic camera can move between preset positions in addition to movements of the normal in-game view system. The camera system remains under the control of the user all the time except at the beginning of a fly-by. FSW has an auto-look feature to look around corners. This feature will handle occlusion using ray-casting from the camera position. The basic Proportional Controller (PC) is a common camera motion control

system used in commercial computer games. While PC has nice arrival characteristics, it is disadvantaged by abrupt exit characteristics and lag of current camera position behind the moving targets. To solve this, Modified Proportional Controller (MPC) is used in FSW. MPC controls exit characteristics by limiting acceleration and correcting lag by considering the velocity of the target points. FSW uses camera cuts to avoid passing through obstacles and whenever the destination is beyond a preset distance. The FSW has one of the most advanced camera control systems among commercial games. Unique solutions to several camera control problems were found, due to special attention given to the issues related to implementation of the camera system during the development of FSW. However, the FSW camera system is affected by more than 100 parameters which pose a tuning problem for the designers and the programmers.

Christie et al. (2005) is a survey on camera control systems. The authors present an overview of automated camera planning techniques, having structured the survey on the expressivity of the set of properties and the characteristics of the solving mechanism. Three levels of image properties are proposed. They are geometric, perceptual and aesthetic. It is observed that the solution technique adopted by the approaches, constrains both geometric abstraction and expressiveness. Christie and Olivier (2006) is an extended survey on camera control systems in computer graphics.

2.5 Constraint Based Camera Control Systems

Badros (1998) is a survey of classes of constraints and their solution techniques used in interactive graphical applications such as drawing, graph layout, visualization and animation systems.

The constraints relevant to the geometric applications are identified as: 1) "linear equalities", 2) "linear inequalities", 3) "linear geometric", 4) "geometric", 5) "geometric in complex plane", 6) "geometric (on camera image)", 7) "point-on-object", 8) "coincident", 9) "arbitrary acyclic", 10) "Horizontal relationship between pairs of points (HOR)", 11) "Vertical relationship between pairs of points (VER)", 12) "Parallel relationship between pairs of line segments (PARA)", 13) "Congruence relationship between pairs of line segments (CONG)", and 14) "Visual Organization Features (VOFs)".

The constraint solving techniques are classified as: 1) "relaxation", 2) "numeric", 3) "iterative numeric", 4) "optimized iterative numeric", 5) "direct numeric (QOCA)", 6) "differential methods", 7) "symbolic", 8) "spring simulation", 9) "graph-layout, direct & iterative", 10) "extreme-bound propagation", 11) "Degree of Freedom (DOF analysis)", 12) "Constraint Logic Programming with Real arithmetic constraints (CLP(R)-like)", 13) "Local Propagation (LP)", and 14) "LP w/o planning".

The constraints relevant to the geometric applications and their solving techniques are shown in table 2. The author classified the constraint solving methods and compared their

expressiveness and performance. A visual classification of interactive constraint solvers is shown in fig. 17. The author observes that 1) the constraints are used to keep the relationship among the on-screen objects, 2) the constraints are used as a declarative means to specify the relationship that the designers or users wish to hold true, 3) the declarative specification of the desired relationship is the fundamental strength of using constraints, and 4) the backtracking algorithm is not successfully used to find solutions in the interactive graphical applications. The author claims that expansion of the classes of constraints used in an application is a significant challenge because the classes of constraints used by the application are restricted by its constraint solver.

Table 2: List of Constraints and Solvers in Interactive Graphical Applications. (Ref: table 1; pp. 3 of Badros 1998)

	System	Author (Year)	Constraints supported	Solving technique	Performance
Drawing	Sketchpad	Sutherland (1963)	geometric	LP, relaxation	$O(n)$, $O(n^2)$
	IDEAL	Van Wyk (1982)	geometric in complex plane	LP w/o planning	$O(n^2)$
	Juno	Nelson (1985)	CONG, PARA, HOR, VER	iterative numeric	$O(n^3)$
	Juno-2	Heydon and Nelson (1994)	CONG, PARA, HOR, VER	optimized iter. num.	$O(n^2)$
	Briar	Gleicher and Witkin (1994)	points-on-object,coincident	differential methods	$O(n^2)$
	Unidraw	Helm et al. (1995)	linear (in)equalities	direct numeric (QOCA)	$O(n^3)$, $O(n^2)$
	GCE	Kramer (1992)	geometric	DOF analysis	$O(ng)$, $O(n \log n)$
	Chimera	Kurlander (1991)	geometric	symbolic, numeric	$O(n^2)$
	Pegasus	Igarashi et al. (1997)	geometric	CLP(R)-like	$O(2^n)$
Graph	CLIDE	Ryall et al. (1997)	VOFs	spring simulation	polynomial
	CGL	He et al. (1996)	linear (in)equalities	iter. numeric	>1 sec (tree $n = 16$)
Visualization	TRIPN, IMAGE	Takahashi et al. (1998)	linear geometric	graph-layout, direct & iterative	"needs to be faster"
	ICOLA	Oster & Kusalik (1998)	linear inequalities	extreme-bound propagation	$O(n + v)$
	Penguins	Chok & Marriott (1998)	linear (in)equalities	direct numeric (QOCA)	interactive ($n \leq 700$)
Animation	TLCC	Gleicher & Witkin (1992)	geometric (on camera image)	differential methods	$O(n^3)$, non-interactive
	Animus	Duisberg (1987)	arbitrary acyclic	LP, relaxation	$O(n)$, $O(n^2)$
	JIM, Parcon	Griebel et al. (1996)	linear (in)equalities, geometric	iterative numeric	<1 sec ($n \leq 100$)

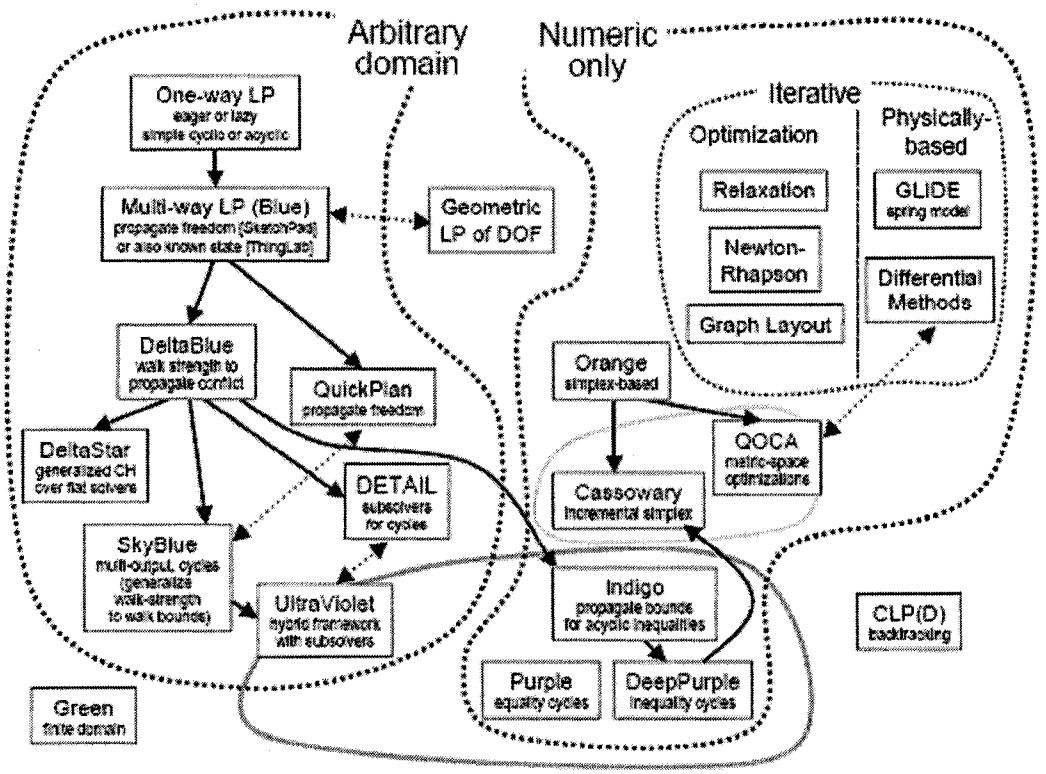

Figure 17: Classification of Interactive Constraint Solvers.
(Ref: fig. 1; pp. 10 of Badros 1998)

Gleicher and Witkin (1992) introduce a camera control system called 'Through-the-Lens Camera Control'. This method enables users to manipulate a camera in a 3D world by controlling and constraining features of the generated image. Camera parameters are not used as camera control parameters (called controls) in this method. Thus, the camera control system can be based on any underlying camera parameterization model. A quaternion based camera model is implemented. Given time derivatives of the camera control parameters, the time derivatives of the camera parameters will be calculated. The problem of calculating time derivatives of the camera parameters is formulated as a simple constraint optimization problem. After finding the solution of the constraint

optimization problem, the camera parameters, over time, are calculated by solving a relevant first-order ordinary differential equation from initial values.

The CamDroid camera control system (Drucker and Zeltzer 1995) consists of a general interpreter, a built in renderer, an object database, and a network of camera modules along with application-specific processes/object interfaces and camera interfaces. To solve the problems of devising a general procedure in a procedural framework for camera control called CINEMA system described in Drucker, Galyean and Zeltzer (1992), Drucker and Zeltzer formalize a constraint based framework. Drucker (1994) deals with the formulation of a consistent framework for camera control across applications from interactive domains classified as exploration, planning, rehearsal, and control using constraint optimization techniques. They propose a design methodology for the framework to control the camera in a 3D virtual environment. The framework is based on the notion that a camera is controlled for particular reasons which can be represented as constraints on the camera parameters. At first, the task requirements for a specific environment are to be analyzed. Then, the camera modules are built with constraints identified from the task requirements. A network of camera modules implements branching conditions between the modules. A camera module consists of an initializer to deal with the changing conditions in the environment over time, a controller to handle user control, a list of constraints, and a local state vector to store camera parameters. The active camera module handles all user inputs. Constraints or camera primitives can be derived from diverse disciplines such as physical limitations of a camera, human factors, cinematography, task performance, and interface design principles. The constraints can

be direct constraints on DOFs in relation to the environment, direct constraints on DOFs in relation to a single object, constraints on derivatives of DOFs, projective constraints based on projection of an object, constraints based on the entire environment, changing constraints, and constraints based on another camera. The constraint list of a camera module can be derived from any of the above mentioned constraints or a combination thereof. A constraint solver finds the final camera parameters for a camera module by satisfying all hard constraints and by optimizing object functions for soft constraints of the module. The solver is implemented in the Feasible Sequential Quadratic Programming coded in C (CFSQP). Five examples of camera frameworks are implemented: a Mission Planner and a Visibility Assistant (Drucker 1994), a Virtual Museum (Drucker 1994; Drucker and Zeltzer 1994), filming conversation between two actors (Drucker 1994; Drucker and Zeltzer 1995), and a Virtual Football Game (Drucker 1994; Drucker and Zeltzer 1995). The authors claim that special constraints can be designed and combined with other constraints by using their method for camera control.

Christianson et al. (1996) formalize a declarative language called Declarative Camera Control Language (DCCL) to encode cinematographic idioms into camera control systems to enhance story telling capabilities of interactive 3D computer graphics applications. Four basic primitive concept or construct of DCCL are fragments, views, placements, and endpoints. Constructs such as shots and idioms are defined by using the primitive constructs. A Camera Planning System (CPS) is designed to codify idioms and to implement the camera placements using idioms for simple movement sequences involving one or two actors. CPS consists of a Sequence Planner, a DCCL Compiler, and

a Heuristic Evaluator. CPS uses an idiom database and a data structure called Film Tree, which consists of the root (Film), internal nodes (Sequence, Scene, and Candidate Idioms), and leaves (Candidate Frames). Input of CPS is trace from an interactive game. CPS finds camera placements based on cinematic principles for a new film to be created from the trace. The Sequence Planner divides the trace into sequences, with each sequence being divided into scenes based on the activities performed in the sequence. The DCCL complier solves DCCL constraints and calculates camera movements. The Heuristic Evaluator ranks the candidate shots and selects the best idioms for each scene of the new film. The system is tested with a simple interactive game consisting of two actors.

Bares, Grégoire and Lester (1998) propose a real-time camera visualization interface system for dynamic 3D world called CONSTRAINTCAM, a real-time 3D camera planning system. The system is extended in Bares and Lester (1999) with multi-shot visualization capability. The proposed intelligent visualization interface uses a partial constraint based framework. The main modules of the extended system are a Constraint Analyzer, a Constraint Solver, and a Multi-Shot Frame Composer. The visualization goal of a user is expressed as a constraint problem. Viewers can select which objects to view and how, specify constraints for each object, and select cinematic style or pace, etc. Given the viewing goal, the Constraint analyzer will determine a consistent region where all the constraints will be satisfied. The Constraint Solver will search the consistent region to find a camera placement that will satisfy all the constraints. If such a solution is not possible, the Constraint Solver relaxes the pairs of weakest incompatible constraints

progressively from a generated incompatible constraints pair graph. If a solution is not found, even after the constraint relaxation process, the problem is decomposed to find a multi-shot solution, in which case, the Multi-Shot Frame Composer creates a sequential or composite visualization solution. This method uses the hierarchical constraints solving strategy (Christie et al. 2005). An example interaction, consisting of multiple autonomous characters interacting in a dense cityscape, is implemented. The real-time performance and results of an informal focus group study with viewers have been reported as encouraging. The authors claim that by using constraints relaxation and creating multi-shot solutions, the system can compute near-optimal solutions to difficult problems.

Jardillier and Languénou (1998) propose a virtual cameraman to help graphic artists create camera movements easily, while, at the same time relieving the artists from having to understand the mathematical concepts underling the camera movements. The virtual cameraman generates a set of camera movements in a dynamic environment which satisfies the constraints, allowing the user to select a series of camera movements to create a desired animation. The solution technique of the virtual cameraman is on a global instead of a per-frame basis. It does not use key framing. The solution set is for the complete animation. Users can define constraints either on the image space or on the object of the scene, or on both. The constraints can be defined for any arbitrary duration of animation because time is considered a variable. A global solving process computes a solution set of camera movements which satisfies multiple constraints for the complete animation. The constraint solver is based on interval arithmetic. It recursively evaluates

the current interval function on hyper-rectangles using tri-valued logic. If the evaluating response for a hyper-rectangle is true, it is a solution and will be added to the solution set. If the evaluating response for a hyper-rectangle is false, sub hyper-rectangles of the hyper-rectangle will not be evaluated further. If the evaluating response for a hyper-rectangle is unknown, the hyper-rectangle is subdivided into two sub hyper-rectangles which will be similarly evaluated further. The authors claim that it is the first method that can compute a complete set of camera positioning solutions.

Languénou et al. (1998) extend the method of Jardillier and Languénou (1998), by using more sophisticated constraint solving techniques implemented using high-level programming in a CLP language. Like a user of the first implementation in Jardillier and Languénou (1998), a user of the second implementation will be able to define constraints on the image space, on the objects of the scene, or on both. The authors use Declarative Language with Interval Constraints (Declic) to utilize two solvers for enforcing hull-consistency over primitive constraints, and box-consistency over global complex constraints. The more sophisticated constraint solving techniques are employed to handle non-linear constraints aroused from modeling camera movements and screen-space constraints. Interval computation is used in the solver to avoid key-framing and interpolation. An extension of the core local-consistency algorithm is used to process universally quantified time constraints. Using the second implementation, satisfaction of the specified constraints can be characterized precisely.

Amerson and Kime (2001) propose a real-time camera control system called Film Idiom Language and Model (FILM) that uses cinematographic techniques and inputs from a Narrative Planner to constraint camera location and orientation in a virtual world of the interactive narratives. Cinematographic idioms will be defined as constraints to find an appropriate camera placement for any shot. Due to the utilization of film idioms, the camera position and movement conveys narrative information along with the events of the narratives. But the exact flow of action will not be predefined in interactive narratives. The user can substantially change the flow of action at any time. The flow of action in the interactive narratives can only be predicted with a high degree of uncertainty. Due to this uncertainty, an automated camera control system in an interactive environment cannot utilize all the idioms of the cinematography. The film idioms are encoded in scene level, a scene tree is constructed using FILM language, and a keyword list is used to search the tree for shot selection. The Narrative Planner generates information based on the actions carried out by the characters and the outcomes of these actions. This information is translated by a domain specific Translator. A domain independent Director uses the translated scene requirements to select a film idiom from the scene tree. It bounds any unbounded variables in the scene to the virtual world objects. These scene specifications, along with the constraints, are used by a graphics engine specific Cinematographer to choose the best camera position to satisfy these constraints. If the selected optimal position is not a valid position, constraints are relaxed based on the weights defined in the scene specification using an ad hoc procedural model to find a point closest to the optimal position which satisfies the remaining constraints. The authors claim that, after implementation of the method, it will be able to select

abstract film idiom knowledge and convert that knowledge into virtual camera placements.

Planning camera shots automatically in virtual 3D environments such that each shot will communicate some specific visual message or goal is dealt with in Bares, Thainimit and McDermott (2000). The authors propose a constraint weighting based camera planning method for 3D virtual environments. A user or a software system requests the visualization of subjects of interest and how they should be viewed. The view request will be translated by a constraint script into a set of constraints. The constraint solver tries to find a solution camera shot based on the given constraints. A solution contains values for camera position, aim direction, up vector and FOV. It uses an exhaustive generate-and-test method to determine the camera placement with the highest cumulative constraint satisfaction rating. The cumulative constraint satisfaction rating is the weighted sum of all individual constraint satisfactions measured as a product of relative priority and satisfaction rating of the constraint. An example application for an over-the-shoulder shot is implemented. The authors claim that the constraints can be used to create cinematic camera shots and to communicate a specific visual message.

In an extended work of camera shot planning, Bares et al. (2000) propose a system consisting of a graphical interface called the Storyboard Frame Editor and a recursive heuristic constraint solver. The frames drawn on the Editor are automatically encoded into a set of constraints. The Editor exports files in two forms of shot definition, namely, the relative displacement definition and the constraint definition. The solver uses the

relative displacement definition for an instantaneous solution if configurations of scene objects match with the 3D scene model of the Editor. Otherwise, the solver uses the constraint definition and finds a solution using an exhaustive generation-and-test method. At first, it finds a valid region for each constraint. Then, it examines candidate camera shots by discretely incrementing camera parameters inside the respective valid regions. Each camera placement is evaluated with the help of an individual constraint evaluation function to find an individual constraint satisfaction rating. The cumulative constraint satisfaction rating is the weighted sum of the individual constraint satisfaction ratings. The search begins at a relatively coarse resolution to find pre-defined numbers of the best candidate shots based on cumulative satisfaction ratings. The search space around the best candidate shots are recursively searched with an increasingly refined resolution. If no camera position is found which satisfies all the constraints, the solver reports failure. Four example camera shots are implemented. The authors claim that the Storyboard Frame Editor and the constraint solver, together, are tools to define how the virtual camera should film objects of interest without worrying about the placement of the camera.

Halper, Helbing and Strothotte (2001) use a predictive camera planning system which obtains a balance between optimality of camera setting and smooth transition with appropriate cuts. All shots in interactive applications are presented to the user immediately without editing. Hence, to plan shots and transitions, the director module should anticipate what is going to happen next. If a good camera position cannot be found in the scene evolved up to that moment due to geometric constraint, a less satisfactory

camera position has to be used. The camera module of the method consists of a Director and a Predictive Camera Planner. Based on past trajectory and acceleration, scene and object states at a future time are calculated by the Planner. The current trajectory is modified by the Planner so that the camera will be at the calculated position at the correct time. It achieves frame coherence by computing a new camera state using existing camera states. The camera position for the next frame is estimated along the path. Then it requests the Director for setting for that future time. The events created by the action module of the game pipeline are used by the Director to select templates from its Shot Library with the help of its Template Selector. To fit successive shots together, various transition rules are utilized to select templates. The Director selects constraints using the templates from its Shot Template for the Constraint Solver of the Planner, and uses its Emotion Template to influence the result of the camera shots. The Constraint Solver is then applied by the Planner on the estimated position to find the camera state for the next frame. An ad-hoc incremental solver has been used for constraint solution (Christie et al. 2005). The Constraint Solver finds a solution for the constraints in hierarchal order in such a way that successive constraint solution influences previous solutions minimally. Each constraint has an optimal setting, a tolerance region, and a relaxation parameter. The relaxation parameter will determine how far the camera can be placed from the optimal value. The camera is placed at the boarder of the tolerance region of a constraint if it lies outside of the region. The system supports the seven constraints, namely, "Height angle", "Level at", "Facing", "Angle to line of interest", "Size", "Visibility", and "Look at". Constraints can be chosen in various combinations. An exploration template was implemented using four constraints namely 'Level at', 'Size', 'Visibility', and 'Look at'.

With this constraint set, the camera should follow the character at its height and keep it in constant view by avoiding obstructions. Three scenarios – a helicopter flying through a city, a human exploring the inside of a medieval building, and a bee flying through a highly cluttered attic – were explored using the template. The authors observed that the camera performed remarkably well in all cases as it avoided obstructions and collisions, and was able to adjust appropriately. They claim that their camera system is the first frame coherent constraint based camera engine and produces a "nearest-best-fit" frame coherent camera solution in real-time by considering future conditions.

A high level modeling approach to camera control is introduced in Christie, Languénou, and Granvilliers (2002), and Christie and Languénou (2003) to find camera parameters to ease the camera path creation and to give the user a declarative tool for animation modeling following the approach of Jardillier and Languénou (1998). The hypertube model to compute camera path for animation is used to avoid expensive computation, camera movement restrictions, and poor interactivity. It provides a global control over the camera path and fine control over the image. Each property of the desired image is defined by a set of constraints over the camera parameters based on established cinematographic techniques. To find a camera path to visualize shots, a dynamic system using a complete search method based on interval filtering is used. A solution will satisfy all the constraints. The paths are based on user-defined sequences of parameterized elementary camera movements. These cinematic primitives are called hypertubes. The hypertubes are defined for linear movement (traveling), rotation around the camera's horizontal or vertical axes (panoramic), zoom-in, zoom-out, and arcing to turn around

objects. The hypertubes are connected by intertubes which are relations between two successive hypertubes that guarantee continuity of camera movement. A hypertube with a set of properties and duration is a Numeric CSP (NCSP). Each cinematographic property is redefined as constraints on the hypertube variables. To model a shot, a user, like a director of a film, provides properties like orientations or locations of the objects on the screen as image and a sequence of elementary camera movements. Then interval consistency and quantified constraint based solving algorithms will compute the parameters of the hypertubes. The search algorithm combines constraint propagation and enumeration to compute a solution of the global NCSP consisting of individual hypertube NCSPs. The solver uses interval-based filtering to remove inconsistencies and isolate inner boxes for each hypertube. It finds a solution based on optimization criteria from the set of the isolated inner boxes of the given hypertube in order to determine starting conditions for the next hypertube. It uses backtracking techniques to remove inconsistent hypertubes. The test results are reported as very encouraging and, as a consequence, open up the way for real-time application of the approach.

Benhamou et al. (2004) deal with finding an efficient solution of control and motion planning problems which, in turn, will enable a graphic artist to specify the desired camera movements for a shot using cinematographic idioms. The method is an extension of Jardillier and Languénou (1998), and Languénou et al. (1998). The constraint solver utilized interval arithmetic and local consistency enforcement based on sound numerical methods to prune the search space. Each constraint is considered one by one along with the sets of elements verifying all the constraints considered so far. During application of

the method to the camera control problems, the user defined descriptions for a shot are translated into constraints in terms of camera parameters. The authors observe that the efficiency of their method increased steadily with progressively higher precision.

Lin, Shih and Tsai (2004) present a constraint based event driven camera control system. The camera movements are automatic, i.e., without any user manipulation, based on cinematographic principles. The authors utilize a sequence of shots per cinematic heuristics to capture a scene. The cinematic heuristics considered are 'do not cross the line', 'avoid jump cut', 'use establishing shots', 'let the actor lead', and 'break movement'. The camera modules are included to handle shots in 'fixed', 'track', 'pan', 'point of view', 'over the shoulder', 'close shot', 'gun fight', and 'close fight' situations. The focus of the method is to improve game experience in a playable game by using cinematic camera control. Like the system in He, Cohen and Salesin (1996), the event-driven computer game system consists of a real-time application, a camera control system, and a renderer. The two systems work similarly. The camera control system consists of camera modules and descriptions of shots. A camera module represents a cinematographic shot and solves the constraints defined by the shot. It handles camera placement and transition based on user input, its constraint list, and parameters from the application. The system has eight camera modules to handle any situation. A description of shot is a sequence of shots. It represents a cinematographic scene and is a finite state machine of camera modules. The descriptions of shots are organized hierarchically. If frame coherence is required, the camera position and angle are interpolated. The constraints and constraint solver used are the same as those used in Halper, Helbing and

Strothotte (2001). The authors claim that the camera control system can generate shots automatically and arranges the shots to create a cinematic effect which is suitable for 3D computer games.

Jhala and Young (2005) propose a camera planning system for narratives in virtual environments where decisions are made in three levels: cinematic geometric composition, camera parameter for information communication, and camera shots and transition to maintain rhetoric coherence. As the narrative based virtual environment becomes more complex, the need for effective communication of information increases. A user perceives the virtual environment through a virtual camera. Thus, the camera can be considered a powerful tool of communication for conveying events, mood of the scene, pace or tempo of the underlying narrative, and the relationship between objects within the virtual environment. The camera planning system is similar to the film production pipeline. The film idioms are formalized to communicate effective information and represented as plan operators. The representation of idioms has casual motivation and hierarchy. It allows the system designer to rank candidate shot sequences. The story is represented as a sequence of actions executed in the 3D world. The knowledge base of the discourse camera planner consists of the sequence of actions executed in the 3D world, and the annotations indicating the properties and characters of the story. The planning system incrementally formulates plans by adding steps to the existing plan to satisfy a goal state or a pre-condition of an existing action by the effect of the action, or to refine an abstract action by expansion. It uses causal reasoning, temporal reasoning, and decomposition during the addition of steps. A ranking function with user preference

then chooses the best plan. The discourse camera planner utilizes a library of hierarchical plan operators to generate a camera directive sequence for specific types of action sequences. The camera directives are translated into constraints. Then, a geometric constraint solver, implemented in the underlying game engine, will find a camera placement solution which satisfies the constraints. The authors observe that the approach may not work at real time for large stories because it needs the story details for pre-planning the camera movement.

Bourne and Sattar (2005) deals with implementation of autonomous third-person perspective camera control for computer games with manageable implementation complexity and system usability. The authors use constraint weighting techniques with a constraint set consisting of height, distance, orientation and frame coherence for a three variable CSP. Each of the constraints, except the frame coherence constraint, has two parameters. One is for the desired value, and another is for the weight of the constraint. The weights indicate preferences among infeasible constraints, and are unique and normalized. Different kinds of camera behaviors can be achieved by manipulating the weights. The weights of the constraint set are referred to as the weighting profile. The cost of a constraint is the weighted difference between its desired value and its current potential solution value. The total cost of a solution for a camera position in a frame is the sum of the costs for individual constraints except the frame coherence constraint. An incomplete Stochastic Local Search technique is applied to find the lowest cost solution for the constraints for a frame. The search method will make a pre-defined numbers of moves and return the best solution found so far as the optimal solution. The frame

coherence constraint prevents camera movement if the cost improvement does not exceed its desired value and act like a simple acceleration and deceleration mechanism. A meta-level visibility constraint using ray casing from the target to the search domain is applied to influence the underlying constraint solver for occlusion avoidance. The authors claim that the gap between implementation and usability requirements can be significantly reduced by this technique.

Bourne (2006) analyzes constraints used in various camera control methods. The constraints may be named differently. The summary is shown in table 3. The author models the camera control problem as a reactive over-constrained weighted CSP problem. A constraint set defines a basic spatial relationship between the camera and the target. Each constraint of the set has a weight and/or a desired value. The set of desired values and weights is defined as the camera profile. The camera profile contains the information needed to determine a state of the camera. The total cost of a solution for a camera position in a frame is the sum of the costs for individual constraints. An efficient constraint solver, called the Sliding Octree Solver, is developed by analyzing the problem domain. The solver will search for an optimal camera position. A game called Dwarf is developed to demonstrate the ability of the camera control system to address existing camera control requirements of Interactive Digital Entertainment systems. Modification of Unreal Tournament 2004 is also done to test the efficiency of the solver and to check real-time performance capability of the camera control system in a commercial game environment. The flexibility and extensibility of the CSP representation are demonstrated by incorporating new capabilities such as frame coherence and occlusion avoidance.

Table 3: Common Constraints.
(Ref: table 2.1; pp. 27 of Bourne 2006)

Constraint	Description
Size	The size of an object in the final image, usually specified as a percentage of the screen.
Placement	The placement of an object in the final image, usually specified as left, right or centre.
Vantage Angle	The altitude to view the object from, sometimes represented as height.
Viewing Angle	The angle to view the object from, usually determined with respect to the facing direction of the object.
In View	To ensure that a given object is within the camera view.
Exclude	To ensure that a given object is not within the camera view.
Depth Order	Specific depth ordering of two or more objects, where one must be closer to the camera than the other.
Occlusion	Minimize (or avoid) the amount of occlusion of an object.

Bourne and Sattar (2006) present a method to implement a camera control system with minimum development time. The constraint based camera control system can be used in various situations with different behaviours. The behaviours can be extended by using more constraints. The camera is controlled through a profile mechanism. By using the profile mechanism, viewing properties and camera movement characteristics can be defined separately.

Bourne, Sattar, and Goodwin (2007) describe the application of CSP solving techniques in the design of a camera control system. The constraint solver of the camera system utilizes the spatial structure of the problem to achieve real-time performance. New capabilities can be added to the camera system by using new constraints. A simple game to find lost treasure is developed to demonstrate how to add additional capability. Extensions such as occlusion avoidance, multiple-target visualization, cinematography, and replays are discussed.

2.6 Occlusion Detection Methods

Christie and Olivier (2006) describe occlusion detection methods in camera control systems. The detection methods use the following detection techniques.

1) Ray casting.
2) Consistent region calculation.
3) Rendering the scene in a hardware stencil buffer.
4) Shadow volume based algorithms.
5) Sub-division of space related to total and partial occlusion cones from bounding spheres of the objects.

Bares and Lester (1999) deal with occlusion as a constraint. All the bounding boxes of the potential occluding objects are projected into a sphere surrounding the object of interest. Next, the projections are converted into a global spherical coordinate system. Finally, these are negated to find a consistent region which satisfies the occlusion avoidance constraint for the object of interest.

Bares et al. (2000) also deal with occlusion as a constraint. Two variations of constraint evaluation are suggested. In a ray casting method, the number of hits by rays from the camera position to the midpoint and vertices of the bounding box of a potential obstruction is counted. Then, the number is used to estimate the fraction of object occlusion. In a frame rendering method, both the object and the potential obstructions are rendered into the OpenGL backbuffer. Next, the non-zero pixel having values not equal

to the id of the object of interest is counted. The number is used to estimate the fraction of object occlusion.

Halper, Helbing and Strothotte (2001) introduce Potential Visibility Region (PVR) to deal with occlusion avoidance. The user of the system can set the preference of the regions for the camera movement by shading the regions with brightest to darkest colours. The depth information of all potential occluders is written into a visibility map. The visibility geometry is rendered into a stenciled buffer from the regions with the brightest to darkest colour. The resultant image buffer consists of the brightest coloured regions that pass the depth test. The brightest colours in the image buffer represent the most desirable regions for camera movements where the target will remain visible.

Pickering (2002) deals with occlusion using a shadow volume technique. Occlusion is described as a function of position in space. If a position of a target object is considered as a position of a light source, finding an occluded region becomes finding a shadow volume created by a bounding box of a potential obstruction. The volume excluding all the shadow volumes is the feasible volume for camera positions for an occlusion-free shot. The feasible volume can be found by the intersection of all feasible volumes of all potential obstructions.

Carlisle (2003) uses ray casting to avoid occlusion of the character. Four rays are cast from the current character position to the locations at the sides as well as the top and bottom of the camera. If any ray collides with an object, the camera is impulsed to the

opposite direction of the camera position to the collision point vector. To prevent oscillation and to allow the user to move the camera, even toward an occluded position when all four rays collide with objects, an overall impulse is applied to move the camera toward the character.

Giors (2004) describes FSW camera collision avoidance methods during normal view and flyby. During normal view, several rays are cast from the target position toward the vicinity of the camera position to deal with collision avoidance. The end points of all the rays have the same altitude that is equal to the camera height. If any ray intersects an object, the camera will be move toward the target. The reduction of the view radius will depend on the position of the ray relative to the target and camera line of sight. During flyby, two rays are cast from the target position to the sides of the camera positions to deal with collision avoidance. If a ray intersects an object, the camera position is moved away from the object by applying an impulse.

Lin, Shih and Tsai (2004) use a line intersection test for collision detection. The test line segment runs from the camera position to the bounding sphere of the actor. If a user desires a smooth camera movement, the line intersection test is replaced by a cylinder intersection test.

Stone (2004) uses a sphere intersection test with triangles of objects. In a virtual camera collision model, a sphere with a finite radius is cast along the line segment from the target character to the desired camera position. If the sphere intersects with any object, the

nearest unobstructed point along the line segment from the target is calculated. The camera is moved to that position. In a physical camera collision model, a sphere is swept from the present camera position to the desired camera position. If the sphere intersects any triangle of an object, the sphere is made to slide along the triangle using the remaining length of the desired camera displacement.

Verth and Bishop (2004) derive mathematical equations to determine if there is any intersection. The authors considered the testing base on triangles and bounding objects (BO) such as sphere, capsule (one type of swift sphere), axis-aligned bounding box (AABB), and object-oriented bounding box (OBB). The test criteria for the intersection between a BO and another same type of BO, a ray and a plane, are listed in table 4. The test criteria for the intersection between a triangle and another triangle, a ray and a plane, are also listed in table 4.

Bourne (2006) and Bourne, Sattar, and Goodwin (2007) deal with occlusion as a constraint. Like Carlisle (2003), four rays are cast from the current character position to the locations at the sides, as well as the top and bottom, of the initial search domain. If any ray intersects an object, the solution cost for the positions closer to the occluded area will be increased. Thus, it is expected that the solver will avoid the occluded area as the solution for the next camera position. When all rays intersect, a safe point relative to the target is selected as next the camera position.

Table 4: Intersection Testing Criteria.

BO/Triangle	Intersection with another same type of BO/Triangle	Intersection with a Ray	Intersection with a Plane
Sphere	Distance between the centres of the spheres are less than the sum of their radii	Distance between centre of the sphere and the ray is less than radius of the sphere	Distance between centre of the sphere and the plane is less than radius of the sphere
Capsule	Distance between the line segments of the capsules are less than the sum of their radii	Distance between the line segment of the capsule and the ray is less than radius the capsule	If the end points of the capsule lie on different sides of the plane or distance between an end point and the plane is less than radius of the capsule
AABB	If there is no plane that separates the two AABB in any coordinate direction	If three line segments overlap one another. Each line segments are defined by intersection of the ray and two yz, zx, or xy planes at maximum and minimum x, y, or z coordinate values respectively of the AABB	If the diagonal most closely align with the plane normal crosses the plane
OBB	If length of projection of line segment between the centres of OBBs is less than the sum of the projections of two extent vectors after all vectors transformed into first OBB's local space.	First transforming the ray into the OBB's local space. Then similar to the AABB and ray intersection test	First transforming the plane into the OBB's local space. Then similar to the AABB and plane intersection test
Triangle	If two edges of first triangle cross the second triangle plane and part of the intersecting line segment lies inside the second triangle	If intersection point between plane of the triangle and the ray lies inside the triangle	If two edges of triangle cross the plane

CHAPTER 3 DESIGN AND METHODOLOGY

In this thesis project, we intend to implement a chase camera. In other words, we intend to follow the main character as it wanders around a 3D virtual environment. We want to keep the character in occlusion-free view at all times. The character may move through open, semi-enclosed, or enclosed areas. In response to character movements, the camera will act autonomously and move automatically without the players' manipulations.

We will work mainly in low level camera control. The desired camera position for a frame will be defined by using a set of constraints. The medium and high level controls will be represented by a user selected camera profile. As mentioned earlier, a camera profile is a set of all weights and/or desired values of the constraints. The challenge is to find an optimal camera position for the next frame cast as a weighted CSP. Starting from the current camera position, the CSP solver will find an optimal camera position for the next frame.

Implementing a reactive constraint based third person perspective chase camera control system for computer games has three major components: formulation of camera control problem as a weighted CSP, modification of the Sliding Octree Solver, and extension of the occlusion detection method.

We discuss these components in the next three sections. In section 3.4, we outline the implementation issues of the project.

3.1 Formulation of Camera Control Problem as a Weighted CSP

We follow the weighted CSP formulation of Bourne and Sattar (2006) that uses the initial constraint set described in Bourne (2006) and other constraints for a three-variable CSP. Three coordinates of the camera position, i.e., X, Y, and Z, will be considered as the variables of the CSP. The domains of the variables will define a region in the space where the camera can be positioned for the next frame. The region will be a cube with equal dimensions along each axis and a subset of the 3D environment (Bourne 2006; Bourne, Sattar and Goodwin 2007). The constraint set of the CSP includes height, distance, orientation, frame coherence, and occlusion constraints.

The initial constraint set consists of height, distance, and orientation constraints, described in 3D world coordinates (Bourne 2006). Each constraint of the set has a weight and a desired value, with the weight of a constraint representing its relative importance. The desired values of the constraints define 'the basic spatial relationships' between the camera and the target character that is visualised in fig. 18.

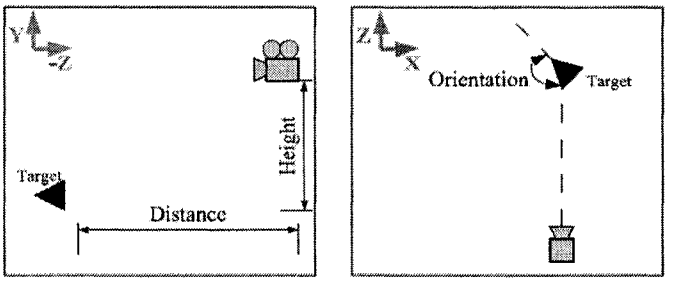

Figure 18: Initial Constraint Set.
(Ref: fig. 3.1; pp. 56 of Bourne 2006)

The weighted CSP solver will evaluate the cost of each constraint for a candidate camera position using the equations shown in table 5. The terms of the equations are self explanatory.

Table 5: Constraint Cost Calculation.
(Ref: eq. 3.2.1, 3.2.2, and 3.2.3; pp. 176 of Bourne and Sattar 2006)

$$Cost_{Height} = | Height_{desired} - \Delta Height | \bullet Weight_{Height}$$

$$Cost_{Distance} = | Distance_{desired} - \Delta Distance | \bullet Weight_{Distance}$$

$$Cost_{Orientation} = | Orientation_{desired} - \Delta Orientation | \bullet Weight_{Orientation}$$

The $\Delta Height$ and $\Delta Distance$ are calculated using the equations shown in table 6. The $\Delta Orientation$ is an angle between the character orientation and the LookAt vectors in degree. The LookAt is a vector from the camera position to the character position. The angle can be calculated using the dot product of the vectors.

Table 6: Real Values of the Constraints.
(Ref: eq. 3.4 and 3.5; pp. 57 of Bourne 2006)

$$\Delta Height = (Camera_y - Character_y)$$

$$\Delta Distance = \sqrt{((Camera_x - Character_x)^2 + (Camera_z - Character_z)^2)}$$

In addition to the initial constraint set, Bourne and Sattar (2006) also use frame-coherence constraint for smooth camera movement, acceleration and deceleration. The cost of the constraint is evaluated using the equation shown in table 7. The frame-coherence constraint has no desired value. It has weight only.

Table 7: Cost for Frame Coherence Constraint.
(Ref: eq. 3.2.4; pp. 177 of Bourne and Sattar 2006)

$$Cost_{Frame\_Coherence} = \frac{|\ Distance_{last\_frame} - Distance_{potential}\ |}{Frame\ interval} \bullet Weight_{Frame\_Coherence}$$

In Bourne and Sattar (2006), occlusion is handled nominally. If the top-middle point of the initial search space is occluded, the solver will change the cost of the height constraint.

The weighted cost for the constraint violation for a camera position is the total cost derived by using the equations for height, distance, orientation, and frame coherence constraints. The total cost also includes the cost change due to occlusion, if any, of the top-middle point of the initial search space for the frame.

3.2 Modification of the Sliding Octree Solver

A typical search tree for a camera control problem, defined as a three variable CSP, is shown in fig 19. The depth of the tree is equal to the number of the variables.

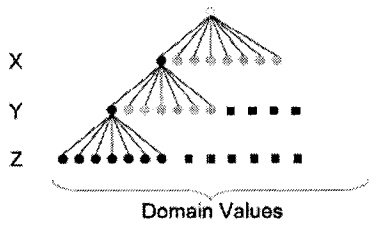

Figure 19: Search Tree.
(Ref: fig. 3.2; pp. 59 of Bourne 2006)

A local search based solver called the Sliding Octree Solver developed in Bourne (2006) and used in Bourne and Sattar (2006) will be used after modification. The Sliding Octree Solver was devised for computational performance. The movement of the search space represented by the Octree data structure is visualized in fig. 20.

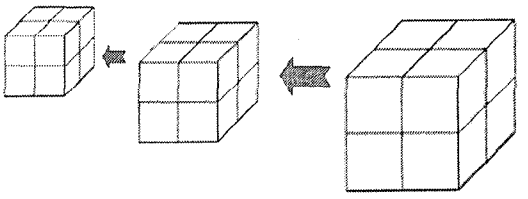

Figure 20: The Octree Spatial Data Structure.
(Adapted from fig. 3.7; pp. 68 of Bourne 2006)

The search tree for the Sliding Octree Solver is shown in fig. 21. The depth of the tree is equal to the number of iterations.

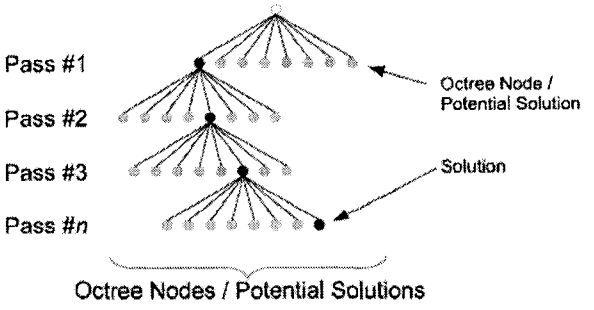

Figure 21: Search Tree for the Sliding Octree Solver.
(Ref: 3.8; pp. 69 of Bourne 2006)

The algorithm of the original Sliding Octree Solver is given in table 8. The Sliding Octree Solver finds a camera position with minimum weighted cost (least constraint violation). At first, the solver divides the current search space into eight regions. Next, it calculates the solution cost at the centres of the regions. It then selects the region having the least cost. It considers the selected region and the surrounding area of the selected region as

the search space for the next iteration. After a predefined number of iterations, it returns the centre of the currently selected region as the optimal camera position.

Table 8: Sliding Octree Solver Algorithm.
(Ref: Algorithm 5; pp. 68 of Bourne 2006)

```
calculate camera facing and right vectors;
calculate initial domain size;
set centre of octree to current camera position;
while current pass less than maximum passes do
   configure octree nodes;
   for each octree node do
      evaluate octree node as potential solution;
      if octree node cost less than best solution cost then
         keep octree node as best solution;
      end
   end
   reduce domain size by scaling factor;
   slide direction = (best solution – octree centre);
   new octree centre += slide direction * domain size;
   increment pass count;
end
return best solution as new camera position;
```

The algorithm for the modified Sliding Octree Solver is shown in table 9. In the modified solver, the desired values of the height and distance constraints of the current camera profile are changed based on the height of the character and the width of the area, respectively, on the fly, whenever the character enters a semi-enclosed/enclosed area. The changes remain effective as long as the character remains inside that area. Moreover, the desired value of the orientation constraint is changed incrementally to find an occlusion-free camera position. All the weight values, however, remain the same. Occlusion testing is done by casting a single ray from the character position to the centre of the current octree region. If the ray is intersected, the centre is an invalid camera position. Camera

Table 9: Modified Sliding Octree Solver Algorithm.

```
calculate camera facing and right vectors;
calculate initial domain size;
set centre of octree to current camera position;
if character is inside an enclosed area do
   change the desired height and distance values;
end
if character comes out from an enclosed area do
   restore the desired height and distance values;
   use camera cut;
end
if character backup to a wall or any obstruction do
   restore the desired height and distance values;
   use camera cut;
end
for each 10 degree increment of desired orientation value do
   while current pass less than maximum passes do
      configure octree nodes;
      for each octree node do
         evaluate octree node as potential solution, if character is visible from the node;
         if octree node cost less than current calculate best solution cost then
            keep octree node as current calculated best solution;
         end
      end
      if current calculated best solution is occluded then
         try next desired orientation value;
      end
      reduce domain size by scaling factor;
      slide direction = (best solution – octree centre);
      new octree centre += slide direction * domain size;
      increment pass count;
   end
   if current calculated best solution is not occluded then
      assume it as the calculated best solution; break;
   end
end
if front or back points at unit distance away from the calculated best solution
                                                        has less cost then
   choose it as calculated best solution;
end
Set best solution in an occlusion free position toward the calculated best solution
                    from the current camera position for smooth camera movements;
restore the desired height and distance values;
return best solution as new camera position;
```

cuts are used whenever the character goes out of a semi-enclosed/enclosed area or the character backs up to a wall or any other obstructions.

3.3 Extension of the Occlusion Detection Method

The occlusion detection method of this thesis project is based on the AABB-ray intersection test technique. For the intersection test of a ray and an AABB, we derive three parameterized intervals $[s_x, t_x]$, $[s_y, t_y]$, and $[s_z, t_z]$ from intersections of the ray and the appropriate AABB planes, as done in Verth and Bishop (2004). The intersections of the ray and the yz planes at minimum and maximum x-coordinates of AABB define the interval $[s_x, t_x]$. Similarly, the intersections of the ray with the zx and xy planes define the intervals $[s_y, t_y]$ and $[s_z, t_z]$, respectively.

The ray is represented by the starting point **P** and the direction vector **v**, as shown in fig. 22. The ray will intersect the yz planes at $x = x_{min}$ and $x = x_{max}$ of the AABB at $R(s_x)$ and $R(t_x)$, respectively. We have

$$P_x + s_x v_x = x_{min}$$

$$P_x + t_x v_x = x_{max}$$

From the above equations, we have

$$s_x = \frac{x_{min} - P_x}{v_x}$$

$$t_x = \frac{x_{max} - P_x}{v_x}$$

Here, we have to consider the following two cases during calculation of the interval.

1. If $v_x = 0$, the ray is parallel to the yz planes. In that case, if $P_x < x_{min}$ or $P_x > x_{max}$, the ray can not intersect the AABB.
2. If both t_x and s_x are less than zero, the ray cannot intersect the AABB.

Similarly, we can derive intervals $[s_y, t_y]$, and $[s_z, t_z]$. During derivation of the intervals, two conditions of non-intersection are also checked. Finally, if there is a pair of non-overlapping intervals, the ray cannot intersect the AABB. If no condition of non-intersection is satisfied, the ray intersects the AABB.

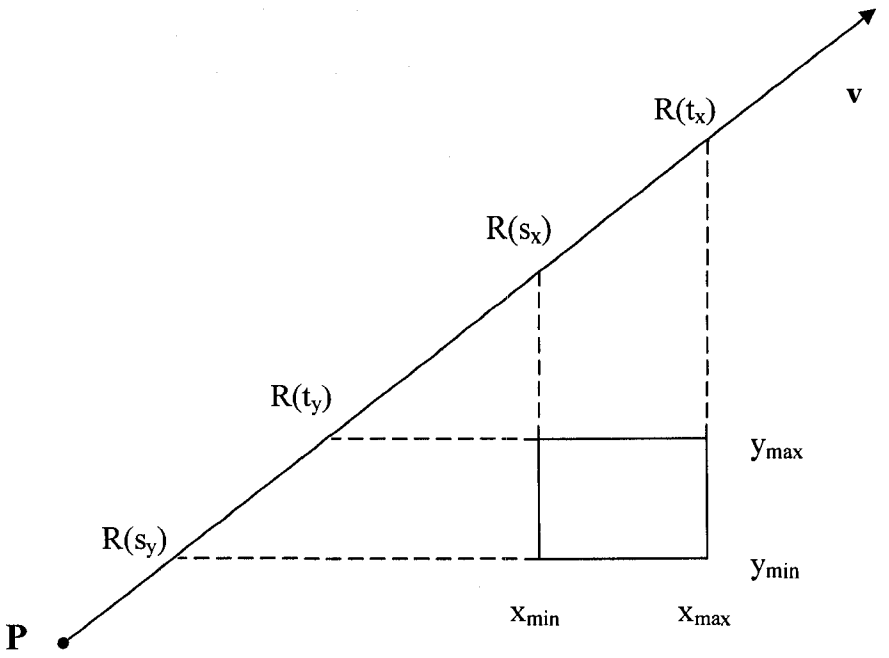

Figure 22: AABB-Ray Intersection Test.
(Ref: fig. 11.14; pp.542 of Verth and Bishop 2004)

The occlusion detection method of Bourne (2006) and Bourne and Sattar (2006) is based on casting four rays from the current character position to four points at the sides, top,

and bottom of the initial search domain for the next frame, and testing the intersection of the rays and the AABBs. The intersection of each ray with each individual AABB of the 3D objects is tested one by one. If the top ray intersects any AABB, Bourne and Sattar (2006) handle the occlusion nominally by changing only the cost of the height constraint. As a result, the technique can not handle occlusion appropriately in all situations.

We have enhanced the occlusion handling capability by using another similar, but separate method. The main occlusion detection method in this thesis project is based on casting another ray from the character position to the centre of the current octree region. Intersection of the ray with each individual AABB of the 3D objects is tested one by one for occlusion detection. The testing is based on the techniques and the code fragments of Verth and Bishop (2004). If the ray is obstructed in any area, the centre of the current octree region is an invalid camera position.

As this thesis project is concerned with following a character which is wandering in open, semi-enclosed, and enclosed areas, we have to consider the view through portals (doors) of the scene objects. We have added the following modifications to the basic AABB-ray intersection testing algorithm to decide if the ray is obstructed in the area under testing.

As mentioned above, a ray is cast from the character to the centre of the current octree region. During intersection testing of the ray with an AABB of an area, five cases may arise based on the positions of the character and the centre of the current octree region.

The five cases are discussed below.

1. **Both the character and the centre of the current octree region are outside of the semi-enclosed/enclosed area. The ray does not intersect the AABB of the area**. The ray is not obstructed.
2. **Both the character and the centre of the current octree region are outside of the semi-enclosed/enclosed area. In addition, the ray intersects the AABB of the area**. If the ray passes through an even numbers of doors, the camera at that position has a clear view of the character. In that case, the ray is not obstructed. If the ray does not pass through an even numbers of doors, the ray is obstructed.
3. **The character is inside and the centre of the current octree region is outside of the semi-enclosed/enclosed area**. If the ray passes through a door of the area, the camera at that position has a clear view of the character. In that case, the ray is not obstructed. If the ray does not pass through a door of the area, the ray is obstructed.
4. **The character is outside and the centre of the current octree region is inside of the semi-enclosed/enclosed area**. Same argument applies as in case 3.
5. **Both the character and the centre of the current octree region are inside of the semi-enclosed/enclosed area**. The ray is not obstructed.

3.4 Outline of the Implementation

This thesis project is implemented in C++ using OpenGL API. OpenGL is a software interface to graphics hardware consisting of approximately 120 distinct commands. Large three-dimensional objects such as automobiles, airplanes or body parts have to be

developed from a small set of geometric primitives such as points, lines, and polygons. It is designed to work efficiently in a client-server environment as a hardware-independent interface. Thus, it has no commands for windowing task or getting user input (Neider, Davis and Woo 1994). To create projection and viewing matrices, the GL Utility (GLU) library will be used. The GLU library is designed to enhance OpenGL capabilities by providing support such as matrix manipulation, polygon tessellation, quadrics, NURBS, and error handling (Chin et al. 1998). In addition, the OpenGL Utility Toolkit (GLUT), a programming interface for windowing task to write system independent OpenGL programs, will be used. GLUT is designed for window system independent interface for OpenGL. GLUT is used to display a graphic scene created by using OpenGL (Kilgard 1996).

Like the camera control system of Bourne and Sattar (2006), the camera control technique developed for this thesis work will use occlusion information from a 3D environment only. Thus, the techniques can be applied to any similar arbitrary 3D environment having open, semi-enclosed, and enclosed areas. Testing will be done for a reasonable amount of time to garner confidence that the developed technique will work for any similar 3D environment.

The outline of the implementation, in brief, is given below.

- The Sliding Octree Solver developed by Bourne and Sattar (2006) is extended to have an unobstructed view of the main character all the time.

- The ability to switch desired distance and height values of the camera profile is incorporated.
- Also incorporated is the incremental change of the desired orientation value of the camera profile whenever the character becomes occluded.
- An occlusion detection method using a single ray from the character to the potential camera position is devised following the approach of Bourne and Sattar (2006) and using the AABB-Ray intersection test.
- Capability to view the main character through the door(s) is added.
- Camera cut is used whenever the main character backs up to walls or any other obstructions.
- Camera cut is also used whenever the main character leaves a semi-enclosed or enclosed area.
- Smooth camera movement during transition is ensured.
- A second camera, to show side views of the main character and the main camera position, is added.
- A third camera, to show top views of the main character and the main camera position, is also added.
- Camera positions for few well-known difficult camera shots from the literature are tested with our implementation.
- We draw our conclusions.

CHAPTER 4 ANALYSIS OF RESULTS

The performance of the reactive constraint based third person perspective chase camera control system for computer games against design goals is evaluated in this chapter. The comparison of performance between the camera control system of Bourne and Sattar (2006) and our camera control system is also done.

A short implementation note is given in section 4.1. Then the test performance of the camera control system in both quantitative and qualitative terms is discussed in section 4.2.

4.1 Implementation

The thesis project is implemented with all the modifications discussed in chapter 3 in C++ using OpenGL API, GL Utility (GLU) library, and OpenGL Utility Toolkit (GLUT) programming interface.

We have also used relevant source codes from the camera control system of Bourne and Sattar (2006). Those codes are for a Windows API application. We have used them subject to required changes for an OpenGL API application.

4.2 Test Performance

We have tested our camera control method in an Intel Centrino Duo, 1.00GB RAM, 2.00GHz, Windows XP Home Edition Version 2002 Service Pack 2 Toshiba Satellite

T2500 laptop. Frame rate was around 60 frames per second. We have also tested our method in a Pentium III, 640MB RAM, 651MHZ, Windows XP Professional Service Pack 2 PC. Frame rate was around 7 frames per second.

Expected or actual camera positions and orientations for different character positions and orientations in various areas are given in fig. 23 to fig. 31. Actual camera positions and orientations for different character positions and orientations in various areas are given in fig. 32 to fig. 36. The actual snapshots are taken by either the second or the third cameras of the project. The selected current camera profile for both the solvers remains the same throughout the testing.

In an actual snap shot, the character is represented by a blue arrow and the main camera under investigation is represented by a smaller red arrow. The arrow heads indicate the orientations. The ground surface of the 3D environment of the project is black with green grid lines. The inside walls of a semi-enclosed/enclosed area in the environment are yellow and the outside walls of the area are dark grey. The floor of the area is light grey.

At the beginning, we compare the performance of the Sliding Octree Solver (SOS) based camera control system and our camera control system with the changing character orientations inside an enclosed area as shown in fig. 23. Then the systems are tested against two well-known difficult situations as shown in fig. 26 and fig. 29. All the figures from fig. 23 to fig. 31 are in top view. The actual snapshots in fig. 24 to fig. 31 are taken by the third camera of the project.

The top views of the expected character and camera positions and orientations while the character rotates counter clock-wise inside an enclosed area are shown in fig. 23. The walls of the area are outlined by the rectangles. The character and camera are represented by the arrow and the circle, respectively. The arrow head indicates the character facing. The character is rotated 45° counter clock-wise from one sub-figure to another.

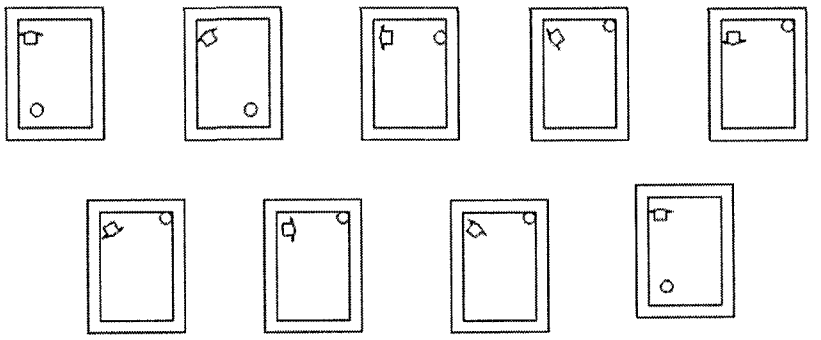

Figure 23: Expected Camera Positions for Different Orientations of the Character.

The top views of the actual camera positions and orientations for the main camera found by the Sliding Octree Solver for different orientations of the main character in an enclosed area are shown in fig. 24. Since the Sliding Octree Solver does not handle occlusion fully, it selects positions at the other side of the walls as valid camera positions for the main camera. The view of an inside character from camera positions outside of the area will be blocked by a wall. Hence, the character cannot be visible at all times.

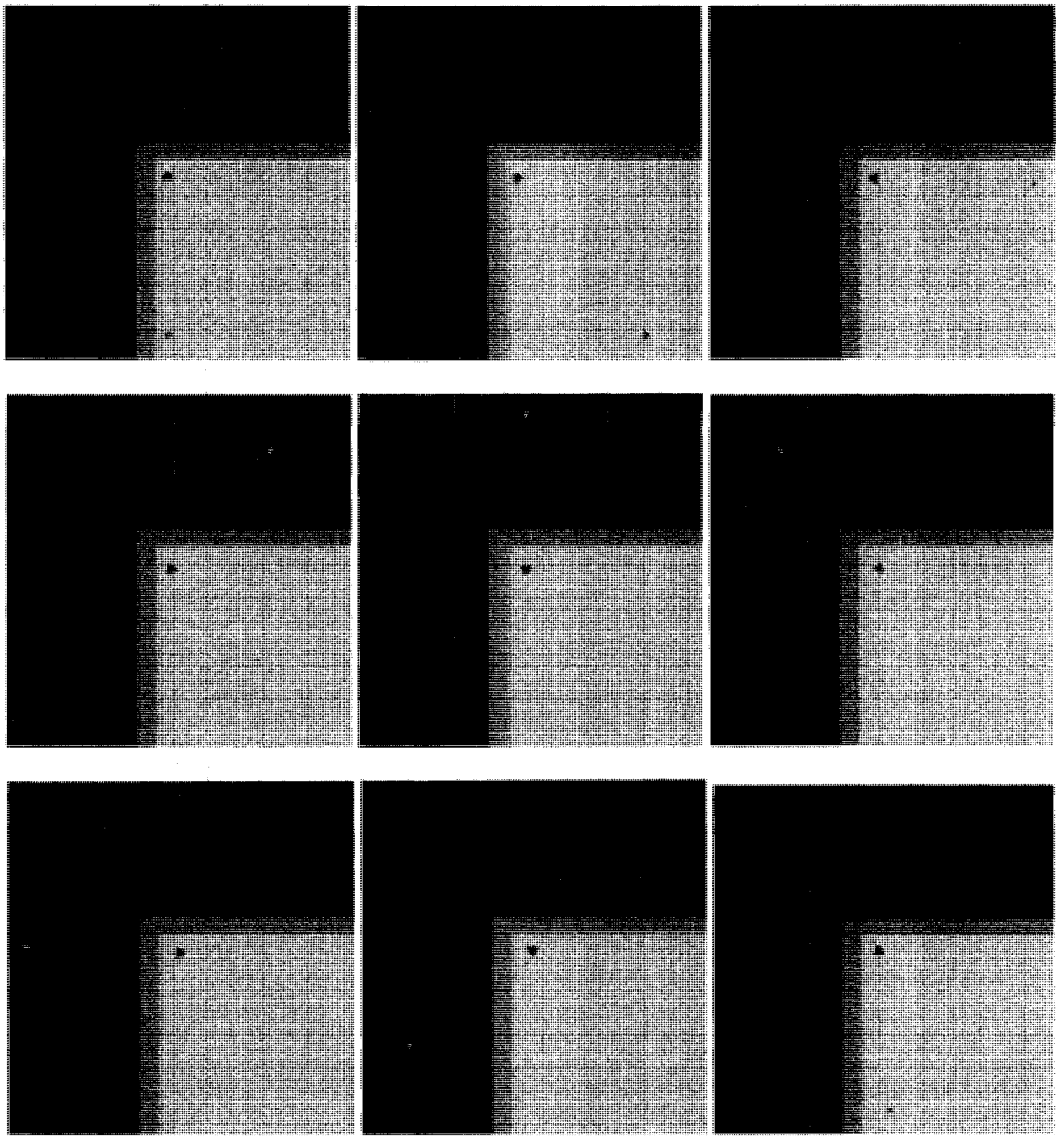

Figure 24: Actual Camera Positions for Different Orientations of the Character by SOS.

The top views of the actual camera positions and orientations for the main camera found by our camera control system for different orientations of the main character in an enclosed area are shown in fig. 25. The actual camera positions in fig. 25 are near the expected camera positions shown in fig. 23. In addition, all the camera positions are

inside the area. The main camera has an unobstructed view of the character all the time. Hence, the character is visible from all positions, resulting in our system (fig. 25) being an improvement over the Sliding Octree Solver (fig. 24).

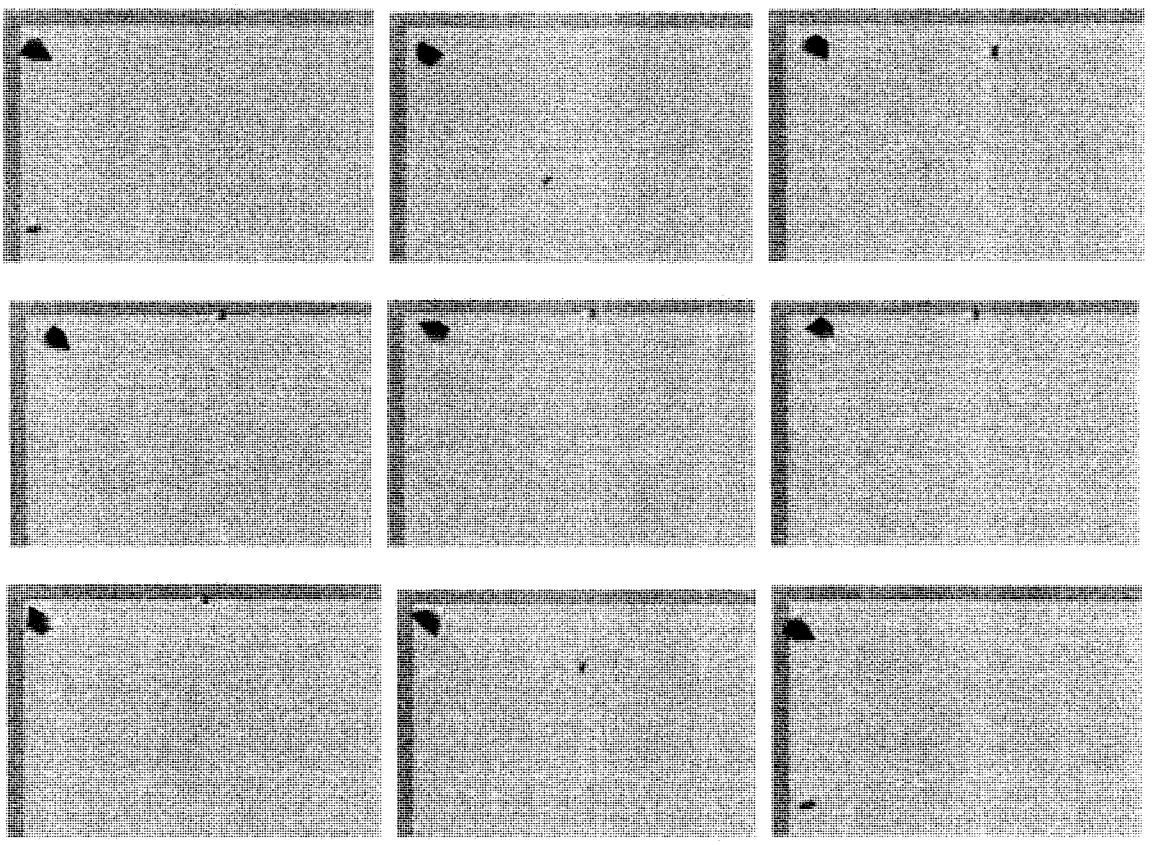

Figure 25: Actual Camera Positions for Different Orientations of the Character.

The top views of the character and camera movements through a crammed doorway are shown in fig. 26. Bold lines outline two rooms and a doorway between the rooms. Character movement is shown in the left sub-figure. The filled circles represent target character positions. In the right sub-figure, expected camera positions for an unobstructed view of the target character are superimposed on the character positions. The circles with the origins represent camera positions. The camera positions are suggested by Stone

(2004). But our requirements are different. In addition to an unobstructed view of the character, the selected current camera profile for both solvers defines an optimal camera position behind the character. Therefore, the actual camera positions are expected to be different.

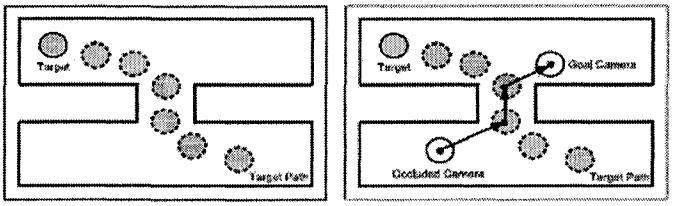

Figure 26: Navigation through a Crammed Doorway.
(Ref: fig. 2.7; pp. 49 of Bourne 2006 which is adapted from fig. 4.1.6 of Stone 2004)

The top views of the actual camera positions and orientations for the main camera found by the Sliding Octree Solver are shown in fig. 27. The Sliding Octree Solver again selects positions beyond the walls as valid camera positions for the main camera. The view of an inside character from camera positions outside of the area will be blocked by a wall.

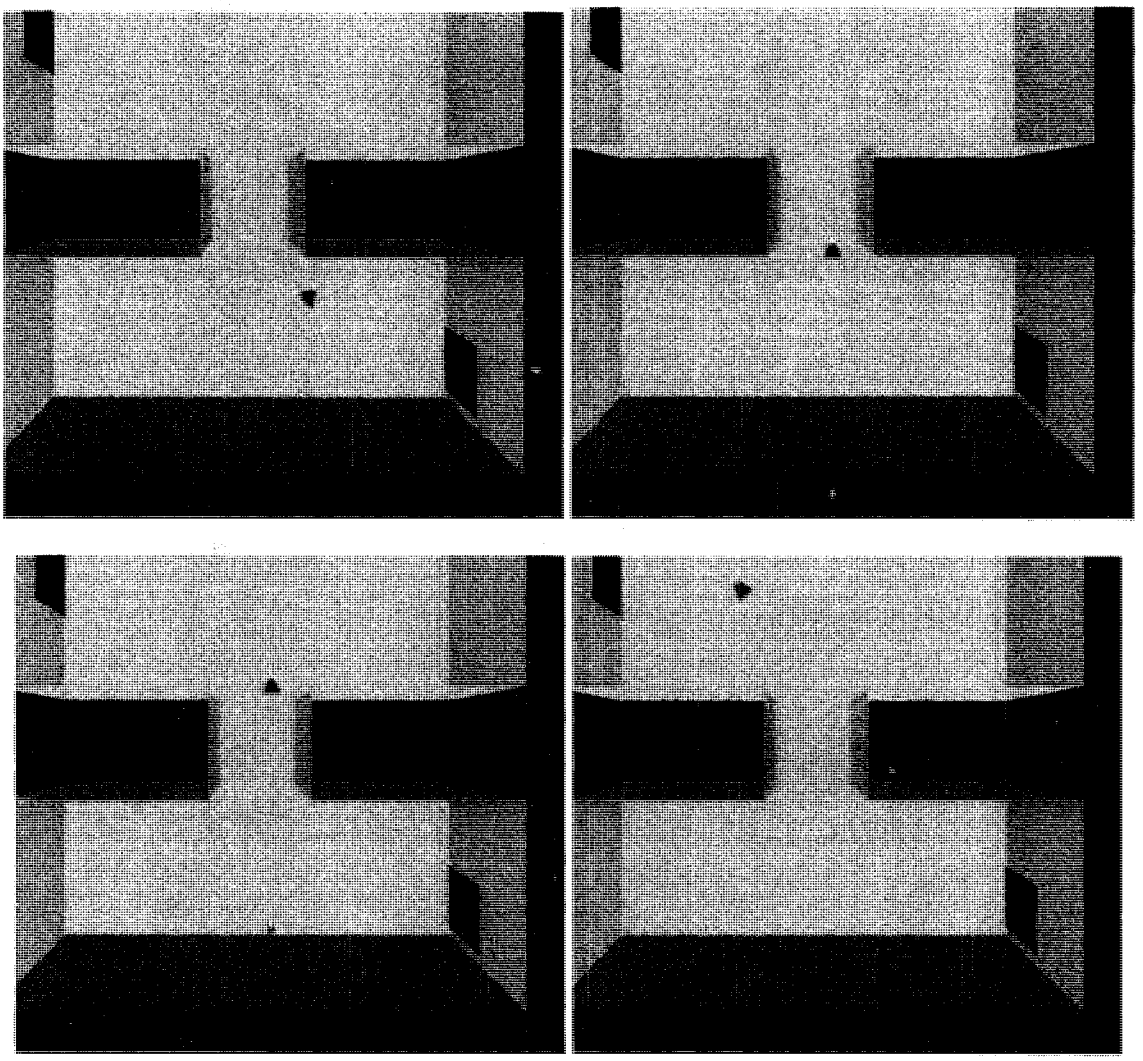

Figure 27: Actual Navigation through a Crammed Doorway by SOS.

The snap shots in fig. 28 show the top views of actual camera positions by our camera control system for different positions of the character in an enclosed area while the character moves through a crammed doorway. Our system was able to handle movements through a crammed doorway nicely. The character remains visible all the time. This is another improvement of our system (fig. 28) compared to the Sliding Octree Solver (fig. 27).

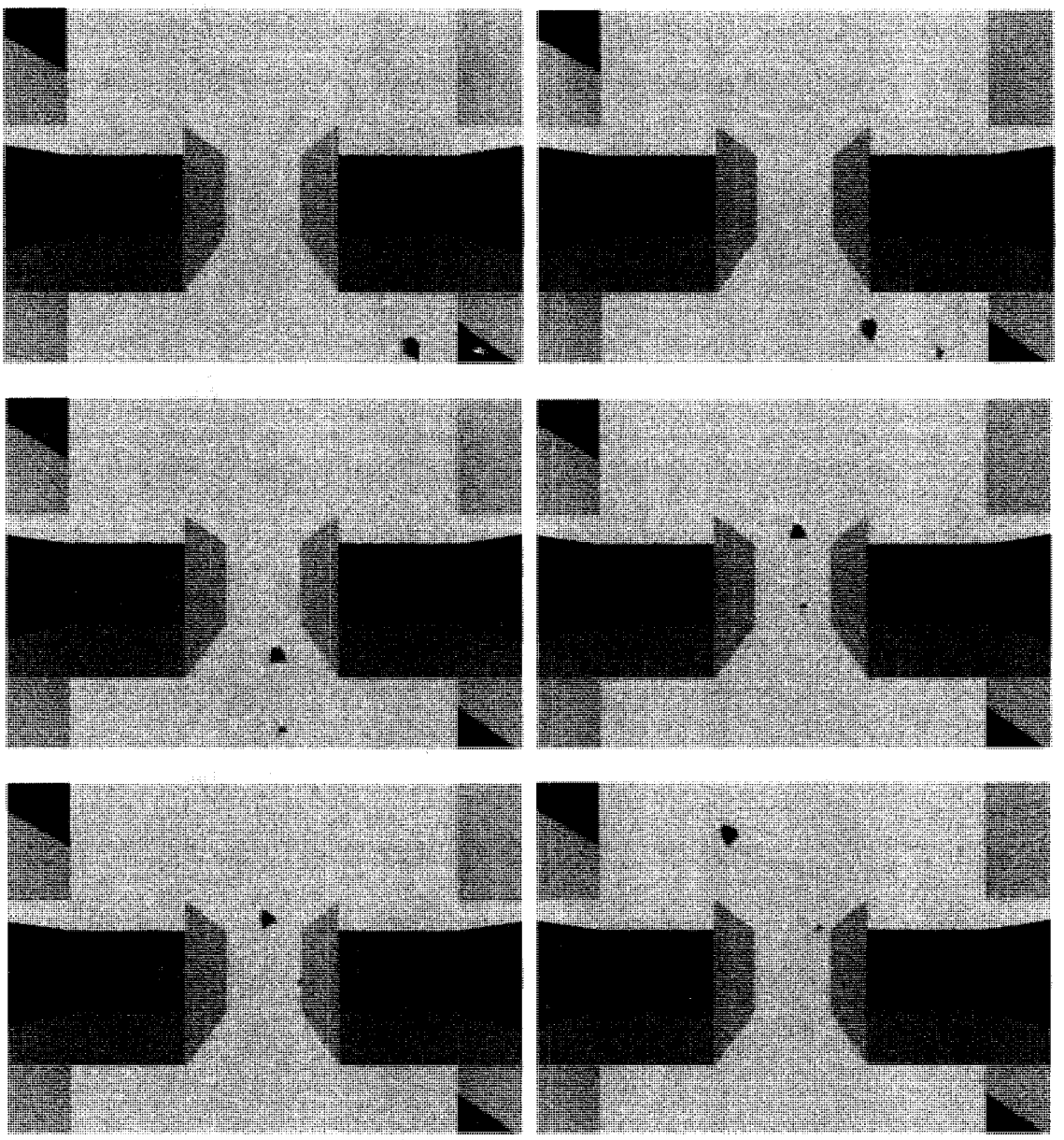

Figure 28: Actual Navigation through a Crammed Doorway.

When the target character enters an enclosed area and moves to a corner of the area, the top views of the expected character and camera positions are shown in fig. 29. The expected final camera position is indicated by P and represented by a circle in the figure. The initial camera position is represented by a camera shape. The target character is represented by a square. The wide straight lines are walls of the enclosed area.

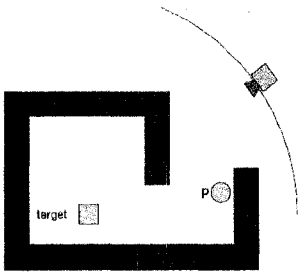

Figure 29: Expected Camera Position when the Character Moves to a Corner. (Ref: fig. 1 of Halper, Helbing, and Strothotte 2001)

The snap shots in fig. 30 show the top views of the actual camera positions for the main camera found by the Sliding Octree Solver when the character enters an enclosed area and moves to a corner of the enclosed area. Here again, the Sliding Octree Solver selects a position on the other side of the obstruction in the right image as a valid camera position.

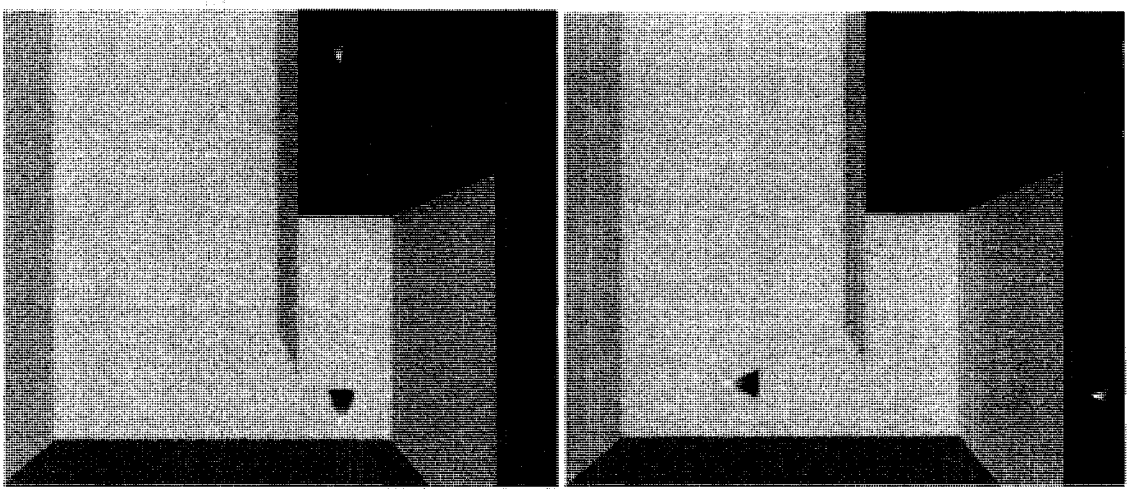

Figure 30: Actual Camera Positions when the Character Moves to a Corner by SOS

The snap shots in fig. 31 show the top views of the actual camera positions for the main camera found by our camera control system when the character enters an enclosed area

and moves to a corner of the enclosed area. Our system is able to find the appropriate camera position in this situation too. This is another improvement of our system (fig. 31) compared to the Sliding Octree Solver (fig. 30).

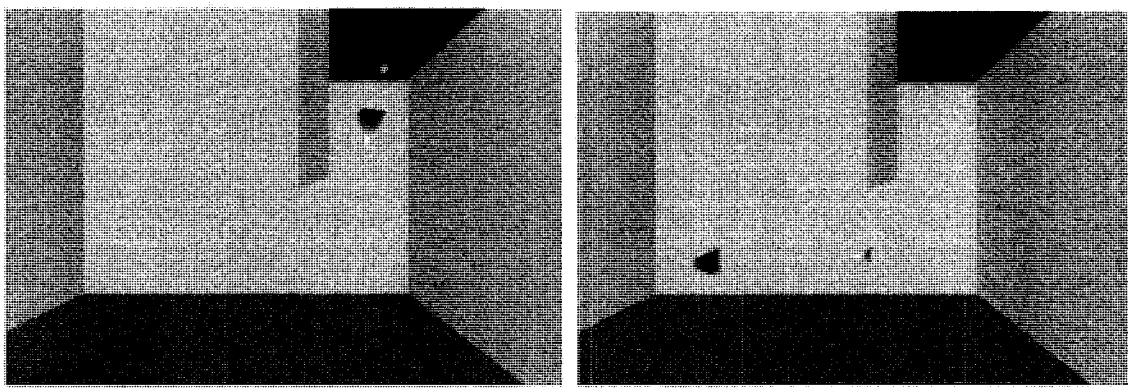

Figure 31: Actual Camera Positions when the Character Moves to a Corner.

As a last comparison, fig. 32 gives another example showing yet another improvement of our system (right image) compared to the Sliding Octree Solver (left image). The snapshots show side views of the actual camera positions taken by the second camera of the project. The character in fig. 32 is inside a room of an enclosed area. The view of the main camera in the left image is blocked by a wall of the room whereas our system, in the right image, finds an obstruction-free position for the main camera inside the room for the same target character (same position and orientation).

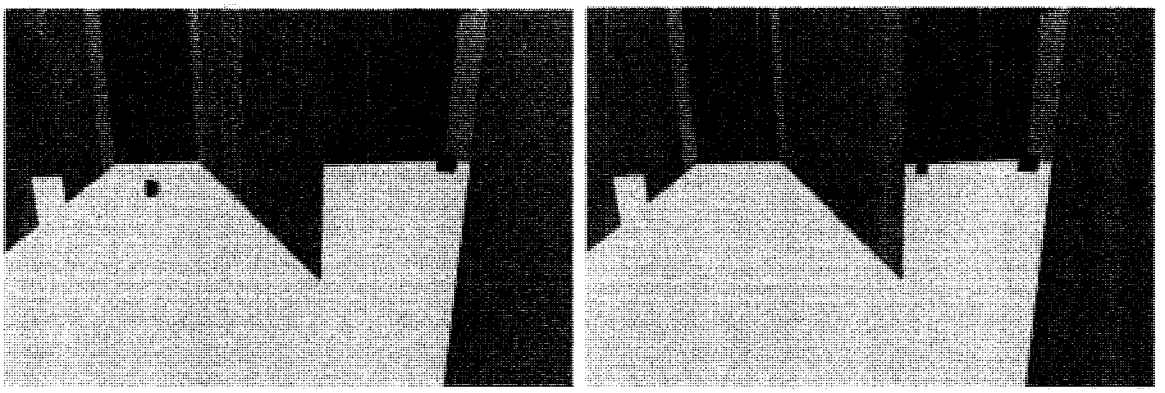

Figure 32: Camera Positions inside an Enclosed Area Found by Two Solvers.

The remaining snapshots show side views taken by the second camera of the project. The next two figures show camera positions by our camera control system during two camera cuts. The camera positions during character movements under an over-hanged obstacle and through a covered pathway are shown in the last two figures.

Most commercial 3D games are not able to handle appropriately the situation whenever the character backs up to a wall or any other obstruction. Some of the methods select a vertical position on top of the character. Others make the obstacle transparent. However, both create a somewhat unnatural feeling. Our system use camera cuts. In fig. 33, the actual positions for the main camera are shown. The positions are found by our camera control method when a character backups against a wall. The result shows a natural camera movement.

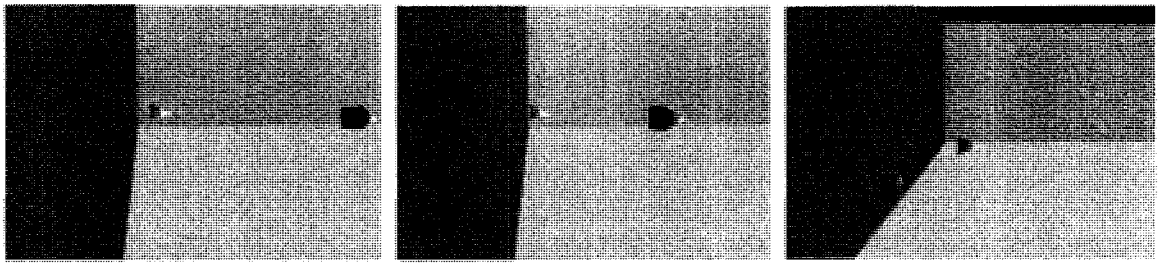

Figure 33: Camera Positions when the Character backs up to a Wall.

When the target character comes out of an enclosed area, our system use another camera cut. The left snap shot in fig. 34 shows the actual position for the main camera found by our camera control system when the character is about to come out of an enclosed area. The right snap shot in fig. 34 shows the actual position for the main camera when the character comes out of the area. The result shows a natural camera movement here too.

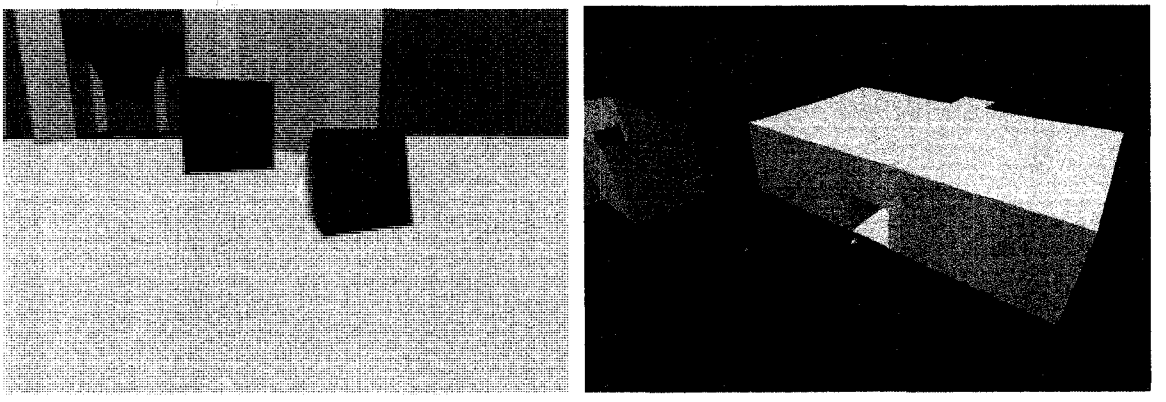

Figure 34: Camera Positions before and after the Character Comes out of an Enclosed Area.

The snap shots in fig. 35 show camera movement parallel to a wall and under a dangled obstacle. Our camera control system is able to find appropriate positions for the main camera to keep the main character visible all the time.

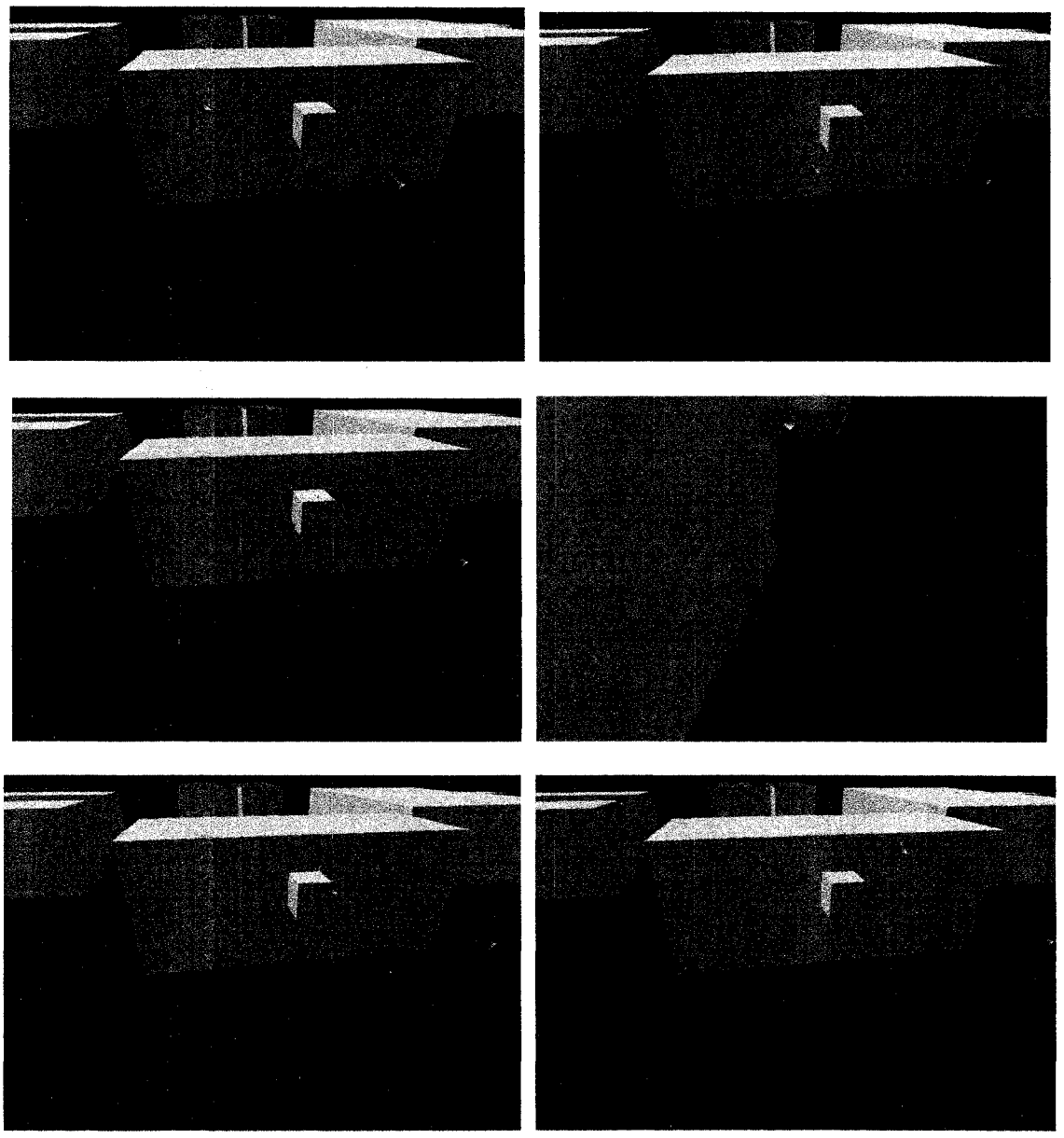

Figure 35: Camera Positions during Movements under an Over-hanged Obstacle.

The snap shots in fig. 36 show the camera positions when the character moves through a covered path way. Our camera control system is also able to find appropriate positions for the main camera to keep the main character visible all the time.

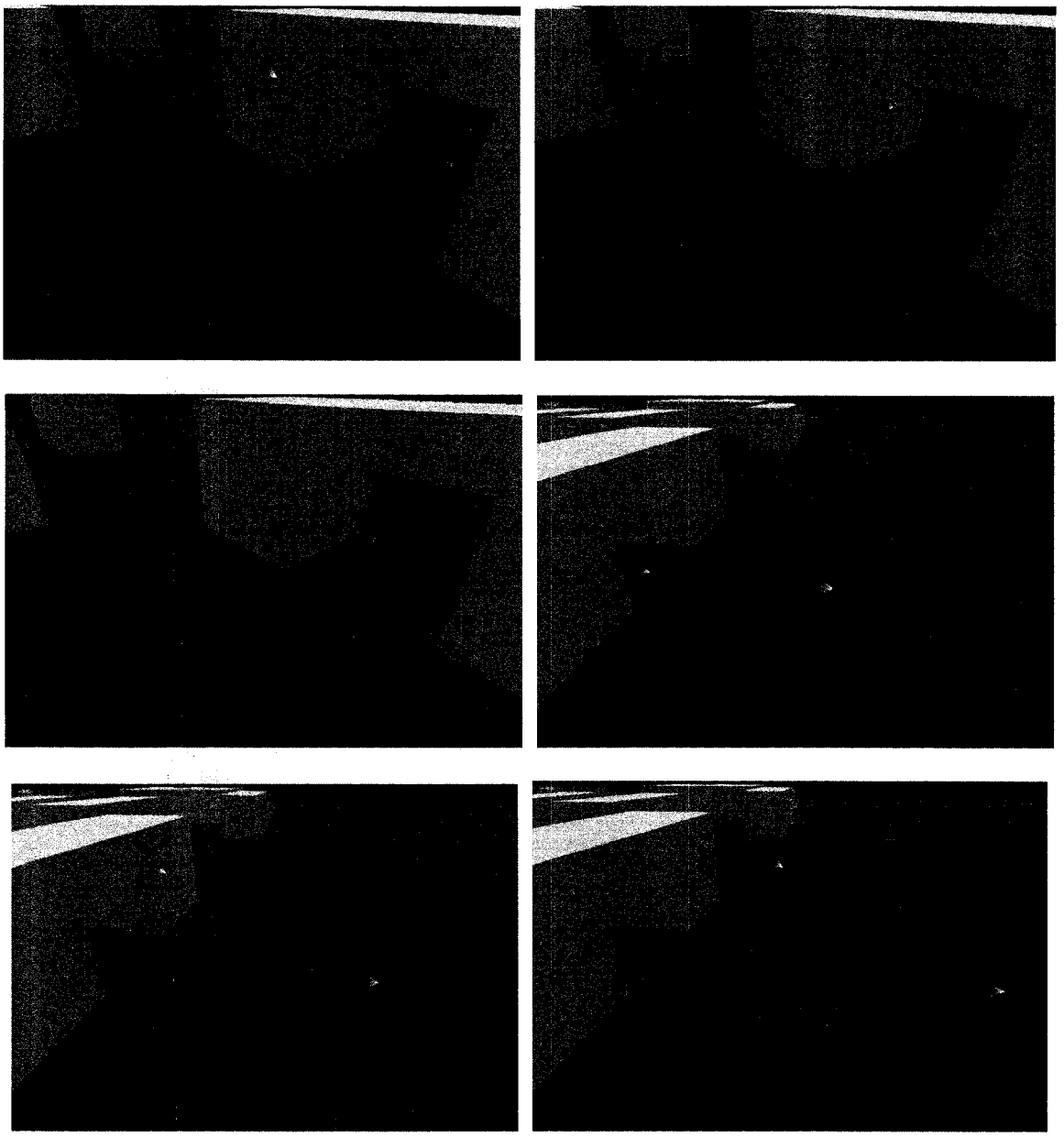

Figure 36: Camera Positions during Movements through a Covered Pathway.

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS

This thesis has described the design, development, and implementation of a reactive constraint based third person perspective chase camera control system for computer games using a local search based CSP solver.

The contents of this thesis are summarized in the next section. In section 5.2, the limitations of the thesis project are discussed. In section 5.3, we draw our conclusions. Finally, future work is discussed in section 5.4.

5.1 Summary of the Contents

Chapter 2 provides a literature review of CSP solving techniques, camera control systems, and occlusion detection systems. Various aspects of the representation of a problem as CSP are discussed. The CSP solution techniques are described and compared. The working principle of a virtual camera, in the context of a graphics pipeline, is discussed with relevant mathematical derivation of transformation matrices and others. General camera control systems and constraint based camera control systems are covered. Finally, occlusion detection systems are discussed.

Chapter 3 provides the formulation of the reactive constraint based third person perspective chase camera control system for computer games as a weighted CSP instance. The objective function, to calculate minimum cost for an optimal camera position, is derived. The modification of the Sliding Octree Solver to enable it to handle

occlusions and to find camera positions for a few difficult situations is discussed. Occlusion detection using the AABB-ray intersection test is covered. Finally, implementation issues of the thesis are outlined.

Chapter 4 provides a performance evaluation of the implemented camera control system against design goals in both quantitative and qualitative terms. Comparative performance analysis with another camera control system based on the original Sliding Octree Solver (Bourne 2006; Bourne and Sattar 2006) is done. A short implementation note is also provided.

5.2 Limitations

The main limitations of this thesis project are listed below.

- The structural entities of the implemented 3D environment consist of static cubic objects only. The objects are made of walls parallel to the coordinate axes. As a result, a single AABB having the same dimensions of an object can fully contain the object. In a more natural scenario, multiple AABBs with different dimensions will be required to represent an object. As a result, the occlusion detection testing becomes less time consuming.
- There is a single moving object in the implemented 3D environment. The target character moves around in the environment. We did not consider any other moving object. As a result, the occlusion detection testing becomes somewhat simplified and less time consuming as well.

- Our camera control system is not tested in a commercial game environment. Thus, performance of the camera system in a commercial game environment is not known.

5.3 Conclusions

The Sliding Octree Solver (Bourne 2006; Bourne and Sattar 2006) is an efficient local search based solver for weighted CSPs. The solver is used in reactive camera planning for real-time performance. We have modified the Sliding Octree Solver, by incorporating the occlusion handling ability fully without compromising the real-time performance of the solver. Our camera can follow a character from the positions defined by a user-specified camera profile.

We have tested our camera control system using some difficult situations. We also compare the performance of our camera control system with another camera control system based on the original Sliding Octree Solver.

Considering all relevant figures in Chapter 4, we find that our camera control system is able to keep the target character visible all the time. The camera control system based on the original Sliding Octree Solver, as shown in fig. 24, fig. 27, fig. 30, and fig. 32, is not able to keep the character visible all the time. This is the main improvement of our camera control system compared to the camera control system based on the original Sliding Octree Solver.

Camera behaviour when the main character backs up to a wall or any other obstruction is a well-known problem faced by many commercial computer games. Many commercial games make the wall/obstacle transparent which creates a somewhat unnatural feeling. Few commercial games solve the problem by choosing a vertical camera position. We have used a camera cut. Our solution is far more natural and works in both open and enclosed areas.

We use another camera cut when the target character comes out of a semi-enclosed or enclosed area. As a result, the user will have a clear view of the character during the change of scenario.

We have also tested our camera control system during character movements under an over hanged obstacle and a covered pathway. The camera was able to keep the character visible at all times during both tests.

Another well-known camera control problem found in many commercial computer games is viewing both sides of a wall or an obstruction with a disturbing feeling. Our technique will allow viewing from one side of a wall or an obstruction only. Implementation of the feature depends on the relevant information from the representation of the virtual environment to make points very close (within 1 pixel) to the walls or obstructions as invalid camera positions. We collected this information during the occlusion detection process.

We have implemented a reactive constraint based third person perspective chase camera control system to follow a character in a 3D virtual world. As the character wanders around the 3D environment from one location to another through open, semi-enclosed, and enclosed areas, the method finds appropriate camera positions in each area keeping the character fully visible at all times. The camera is able to follow the character from the desired height, distance, and orientation defined by the current camera profile. The camera control system has real-time performance.

5.4 Future Work

The direction of future work is given below:

a) To investigate the functionality of the Editor in computer games to establish the relationship of the Editor among the Director and the Cinematographer.

b) To develop two higher level camera control systems on top of the low level camera control system, i.e., the Camera Operator, developed in the present work. A medium level camera control system, i.e., the Cinematographer, will incorporate cinematic idioms to enhance viewers' experience. A high level camera control system, i.e., the Director, will enhance viewers' experience further by incorporating mood, pace, etc.

c) The occlusion detection used in the present work is based on ray intersection with the axis aligned bounding boxes (AABB) of the objects. More comprehensive occlusion detection using binary space portioning (BSP) or other advance data structures could enhance performance in more complex and realistic game world scenarios.

REFERENCES

1. Ali, M. L. and Goodwin, S. D. (2008) Applications of CSP Solving in Camera Control. In Proceedings of *IEEE Consumer Communications and networking Conference* [To appear] [Accepted on September 24, 2007], Las Vegas, NV, USA, January 10-12, 2008.

2. Amerson, D. and Kime, S. (2001) Real-time Cinematic Camera Control for Interactive Narratives. In Working Notes of the *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA, USA, AAAI Press, 1-4.

3. Arijon, D. (1976) *Grammar of the Film Language.* Communication Arts Books, Hasting House, Publishers, New York.

4. Badros, G. J. (1998) *Constraints in Interactive Graphical Applications,* Ph.D. General Examination, University of Washington, Box 352350, Seattle, WA 98195-2350, USA.

5. Bares, W. H., Grégoire, J. P., and Lester, J. C. (1998) Realtime Constraint-based Cinematography for Complex Interactive 3D Worlds. In Proceedings of *Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI),* Madison, WI, USA, AAAI Press, 1101-1106

6. Bares, W. H. and Lester, J. C. (1999) Intelligent Multi-shot Visualization Interfaces for Dynamic 3D Worlds. In Proceedings of *1999 International Conference on Intelligent User Interfaces (IUI'99)*, Los Angeles, California, USA, 119-126.

7. Bares, W., McDermott, S., Boudreaux, C., and Thainimit, S. (2000) Virtual 3D Camera Composition from Frame Constraints. In Proceedings of *Eight ACM International Conference on Multimedia*, Marina Del Ray, Los Angeles, CA, USA, 177-186.

8. Bares, W. H., Thainimit, S., and McDermott, S. (2000) A Model for Constraint-based Camera Planning. In Proceedings of *AAAI 2000 Spring Symposium Series on Smart Graphics*, Stanford, CA, USA, 84-91.

9. Barták, R. (1998) On-Line Guide to Constraint Programming. *http://kti.ms.mff.cuni.cz/ bartak/constraints/index.html.*

10. Benhamou, F., Goualard, F., Languénou, E., and Christie, M. (2004) Interval Constraint Solving for Camera Control and Motion Planning. *ACM Transactions on Computational Logic,* **5(4)** 732-767.

11. Blinn, J. (1998) Where am I? What am I Looking at? *IEEE Computer Graphics and Applications, July1998,* 76-81.

12. Boufama, B. (2007) Lecture Slides in *http://boufama.uwindsor.ca/352/lectures.html for course 60-352 and in http://boufama.uwindsor.ca/551/lectures.html for course 60-551,* University of Windsor, October 21, 2007.

13. Bourne, O. (2006) *Constraint-Based Intelligent Camera Control for Interactive Digital Entertainment.* Ph.D. thesis, Griffith University, Queensland, Australia.

14. Bourne, O. and Sattar, A. (2005) Applying Constraint Weighting to Autonomous Camera Control. In Proceedings of the *First Artificial Intelligence and Interactive Digital Entertainment Conference,* Marina Del Ray, CA, USA, AAAI Press, 3-8.

15. Bourne, O. and Sattar, A. (2006) Autonomous Camera Control with Constraint Satisfaction Methods. In Rabin, S., editor, *AI Game Programming Wisdom 3,* Charles River Media, 173-187.

16. Bourne, O., Sattar, A., and Goodwin, S. D. (2007) A Constraint-Based Autonomous 3D Camera System. *Constraints* [To appear], *http://ai.uwaterloo.ca/~vanbeek/Constraints/constraints.html#forthcoming.*

17. Carlisle, P. (2003) An AI Approach to Creating an Intelligent Camera System. In Rabin, S., editor, *AI Game Programming Wisdom 2,* Charles River Media, 179-185

18. Chin, N., Frazier, C., Ho, P., Liu, Z., and Smith, K. P. (1998) *The OpenGL Graphics System Utility Library (Version 1.3).* Editor (version 1.3), Jon Leech, Silicon Graphics Inc.

19. Christianson, D. B., Anderson, S. E., He, L. W., Salesin, D. H., Weld, D. S., and Cohen, M. F. (1996) Declarative Camera Control for Automatic Cinematography. In Proceedings of *Thirteenth National Conference on Artificial Intelligence (AAAI 96),* Portland, OR, USA, AAAI Press, 148-155

20. Christie, M. and Languénou, E. (2003) A Constraint-Based Approach to Camera Path Planning. In Butz, A., Krüger, A., and Olivier, P., editors, Proceedings of the *third International Symposium on Smart Graphics,* Heidelberg, Germany, LNCS, Springer, **2733** 172-181.

21. Christie, M., Languénou, E., and Granvilliers, L. (2002) Modeling Camera Control with Constrained Hypertubes. In Hentenryck, P. V., editor, Proceedings of the *Eighth International Conference on Principles and Practice of Constraint Programming (CP2002)*, Ithaca, NY, USA, 618-632.

22. Christie, M., Machap, R., Normand, J. M., Olivier, P., and Pickering, J. (2005) Virtual Camera Planning: A Survey. In Butz, A., Fisher, B., Krüger, A., and Olivier, P., editors, *Smart Graphics: 5th International Symposium Proceedings, SG 2005,* Frauenwörth Cloister, Germany, LNCS, Springer, ISBN: 3-540-28179-7, **3638** 40-52.

23. Christie, M. and Olivier, P. (2006) Camera Control in Computer Graphics. In Gröller, E. and Szirmay-Kalos, L., guest editors, *EUROGRAPHICS 2006*, **25-3**.

24. Drucker, S. (1994) *Intelligent Camera Control in Graphical Environments.* Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

25. Drucker, S. M., Galyean, T. A., and Zeltzer, D. (1992) CINEMA: A System for Procedural Camera Movements. In Proceedings of *1992 Symposium on Interactive 3D Graphics,* Cambridge, MA, USA, 67-70

26. Drucker, S. M. and Zeltzer, D. (1994) Intelligent Camera Control in a Virtual Environment. In Proceedings of *Graphics Interface '94*, Banff, Alberta, Canada, 190-199.

27. Drucker, S. M. and Zeltzer, D. (1995) CamDroid: A System for Implementing Intelligent Camera Control. In Proceedings of the *1995 Symposium on Interactive 3D Graphics,* Monterey, CA, USA, 139-144.

28. Giors, J. (2004) The Full Spectrum Warrior camera system. In Proceedings of *Game Developers Conference, 2004,* San Jose, CA, USA.

29. Gleicher, M. and Witkin, A. (1992) Through-the-Lens Camera Control. In Catmull, E. E., editor, *Computer Graphics (Proceedings of SIGGRAPH '92),* **26(2)** 331-340.

30. Halper, N., Helbing, R., and Strothotte, T. (2001) A Camera Engine for Computer Games: Managing the Trade-off between Constraint Satisfaction and Frame Coherence. In *Computer Graphics Forum,* **20(3)** 174-183.

31. Halper, N and Olivier, P. (2000) CAMPLAN: A Camera Planning Agent. In Proceedings of *AAAI 2000 Spring Symposium on Smart Graphics,* Stanford, CA, USA, 92-100.

32. Hawkins, B. (2003) Creating an Event-Driven Cinematic Camera, Part Two. *http://www.gamasutra.com/features/20030110/hawkins_01.htm,* January 10 2003.

33. He, L., Cohen, M. F., and Salesin, D. H. (1996) The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing. In Proceedings of *23rd Annual Conference on Computer Graphics (SIGGRAPH 96)*, New Orleans, LA, USA, ACM Press, 217-224

34. Hornung, A., Lakemeyer, G., and Trogemann, G. (2003) An Autonomous Real-Time Camera Agent for Interactive Narratives and Games. In Proceedings of Fourth International Workshop on Intelligent Agents (*IVA 2003)*, Kloster Irsee, Germany, Springer, 236-243.

35. Huang, M. (2004) *Dueling CSP Representations: Local Search in the Primal versus Duel Constraint Graph.* Masters' thesis, University of Windsor, Windsor, Ontario, Canada.

36. Image_1 (2007) First Person Shooter – World War 2 Online in *http://en.wikipedia.org /wiki/Image:SShot4862.jpg.*

37. Image_2 (2007) Second Person Shooter in *http://www.selectparks.net/ modules.php? name=News&file=article&sid=284.*

38. Image_3 (2007) Third Person Shooter in *http://en.wikipedia.org/wiki /Image:Tomb_Raider_City_of_Vilcabamba.png.*

39. Jardillier, F. and Languénou, E. (1998) Screen-Space Constraints for Camera Movements: the Virtual Cameraman. In Ferreira, N. and Göbel, M., editors, Proceedings of *Computer Graphics Forum (Eurographics-98),* Blackwell Publishers, ISSN 1067-7055, **17(3)** 175-186.

40. Jhala, A. and Young, R. M. (2005) Discourse Planning Approach for Cinematic Camera Control for Narratives in Virtual Environments. In Proceedings of the *National Conference of the American Association for Artificial Intelligence,* Pittsburg, PA, USA, AAAI Press, 307-312.

41. Kilgard, M. J. (1996) *The OpenGL Utility Toolkit (GLUT) Programming Interface (API Version 3).* Silicon Graphics, Inc.

42. Kumar, V. (1992) Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine,* **13(1)** 32-44.

43. Languénou, E., Benhamou, F., Goualard, F., and Christie, M. (1998) The Virtual Cameraman: an Interval Constraint Based Approach. In Proceedings of the *Thirteenth European Conference on Artificial Intelligence (Constraint Techniques for Artistic Applications Workshop),* Brighton, UK.

44. Lin, T., Shih, Z., and Tsai, Y. (2004) Cinematic Camera Control in 3D Computer Games. In Proceeding of the *Twelfth International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2004, WSCG 2004,* University of West Bohemia, Campus Bory, Plzen-Bory, Czech Republic, 289-296.

45. Nadel, B. A. (1988) Tree Search and Arc Consistency in Constraint Satisfaction Algorithms. In Kanel, L. and Kumar, V., editors, *Search in Artificial Intelligence,* Springer-Verlag, New York, 287-342.

46. Nadel, B. A. (1990) Constraint satisfaction algorithms. *Computational Intelligence,* **5(4)** 188-224.

47. Neider, J., Davis, T., and Woo, M. (1994) *OpenGL Programming Guide.* Silicon Graphics, Inc. Addison-Wesley Publishing Company, http://fly.cc.fer.hr/~unreal/theredbook/.

48. Oliveros, D. A. M. (2004) *Intelligent Cinematic Camera for 3D Games.* Masters' Project B2 Report, University of Technology, Sydney, Australia.

49. Pickering, J. H. (2002) *Intelligent Camera Planning for Computer Graphics,* Ph.D. thesis, Department of Computer Science, University of York, York, YO10 5DD, UK

50. Russell, S. and Norvig, P. (2002) *Artificial Intelligence: A Modern Approach.* 2nd Edition, Prentice Hall/Pearson Education, Inc., Upper Saddle River, NJ, USA.

51. Stone, J. (2004) Third-Person Camera Navigation. In Kirmse, A., editor, *Game Programming Gems 4,* Charles River Media, 303-314.

52. Treglia II, D. (2000) Camera Control Techniques. In DeLoura, M., editor, *Game Programming Gems,* Charles River Media, 371-379.

53. Tsang, E. (1993) *Foundations of Constraint Satisfaction.* Academic Press Inc., San Diego, CA, USA.

54. Verth, J. M. V. and Bishop, L. M. (2004) *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide.* Morgan Kaufmann, March 25 2004.

55. Ware, C. and Osbom, S. (1990) Exploration and Virtual Camera Control in Virtual three-dimensional Environments. In *1990 Symposium on Interactive 3D Graphics,* 175-184.

56. Yang, J. (2004) *High Performance Constraint Satisfaction Problem Solving: State-Recomputation versus State-Copying.* Masters' thesis, University of Windsor, Windsor, Ontario, Canada.

VITA AUCTORIS

Mohammed Liakat Ali was born in Bangladesh. He obtained his first degree, a BSc in Electrical Engineering, from the Bangladesh University of Engineering & Technology (BUET), Dhaka in 1978. He worked as an electrical engineer in the field of operation, maintenance, and trouble shooting of electrical systems in electrical sub-stations, power stations, and chemical and petrochemical industries. He also worked in installation, testing, and commissioning of electrical projects. He obtained his second degree, a BCS (Honours) in Computer Science, with high distinction from the University of Windsor in 2004. He is currently a candidate for the MSc in Computer Science degree at the University of Windsor and hopes to graduate in fall 2007.